IBM大学合作项目书籍出版资助

教育部–IBM高校合作项目
精品课程系列教材



大型主机系统管理 REXX编程详解

高珍 主编

刘恒 王天琦 张润芸 庄焕焕 编著

清华大学出版社

IBM 大学合作项目书籍出版资助
教育部—IBM 高校合作项目精品课程系列教材

大型主机系统管理
REXX 编程详解

高　珍　主编
刘　恒　王天琦　张润芸　庄焕焕　编著



清华大学出版社
北　京

内 容 简 介

本书详细介绍了 REXX 的语法、函数和子例程的使用、REXX 与关键子系统的交互、REXX 程序的执行和调试等，并辅以案例和实验，是大型主机专业系统管理方向的重要教材，全书共分 10 章，是一本理论与实践并重的教材。

本书的第 1 章介绍大型主机上常用的几种脚本语言；第 2 章阐述主机脚本语言 REXX 的由来、发展、特点和组成等内容；第 3 章详细介绍 REXX 的基本语法，包括指令、表达式、程序控制流等；第 4 章介绍函数和子例程的使用；第 5 章重点阐述 REXX 数据处理技术，包括 REXX 强大的数据解析功能、数据栈和文件操作；第 6 章介绍 REXX 与主机子系统或工具的交互，包括 TSO、MVS 控制台、JES、ISPF 和 USS 等；第 7 章介绍了 REXX 与 ISPF 的交互；第 8 章介绍 REXX 程序的执行方式；第 9 章介绍 REXX 程序的调试技术；第 10 章介绍 3 个综合案例；第 11 章针对 REXX 主要知识点设计了 REXX 实验，并附有实验答案。

本书可以作为高等院校计算机学院、软件学院有关大型主机课程的教材，也可以作为从事大型主机工作的相关技术人员，尤其是系统管理人员的自学书籍，也可供希望学习和了解 REXX 编程技术的人员参考。

图书在版编目（CIP）数据

大型主机系统管理 REXX 编程详解/高珍主编；刘恒等编著.--北京：清华大学出版社，2012.5
ISBN 978-7-302-28005-7

Ⅰ. ①大…　Ⅱ. ①高…　②刘…　Ⅲ. ①大型计算机－程序设计　Ⅳ. ①TP338.4

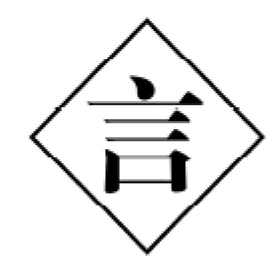
中国版本图书馆 CIP 数据核字(2012)第 019875 号

责任编辑：龙启铭　顾　冰
封面设计：傅瑞学
责任校对：白　蕾
责任印制：何　芊

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：北京富博印刷有限公司
装 订 者：北京市密云县京文制本装订厂
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：14　　**字　　数**：324 千字
版　　次：2012 年 5 月第 1 版　　**印　　次**：2012 年 5 月第1 次印刷
印　　数：1～3000
定　　价：29.00 元

产品编号：043362-01

FOREWORD

对于系统管理员而言，掌握脚本编程技术的重要性不言而喻。不管如今系统架构如何庞大和复杂，系统管理维护的很多日常的基础性工作还是完全可以程式化、机械性地定义和描述，并通过机器加以执行完成的。系统脚本编程不仅仅是将管理员从重复的键盘敲击和命令输入中解放出来，更重要的是最大程度地避免了重复性工作所可能带来的“疲劳性”错误，对于拥有最高系统权限的管理员而言，这些错误很可能是灾难性的。而另一方面，各种脚本语言也从简单的“命令执行序列”发展成为了更加精巧完善且易于掌握的快速开发语言，吸引了更为广泛的关注，并且延伸和扩展到了更为高级和复杂的应用之中。

如同 Perl 之于 UNIX/Linux 一样，对于大型机操作系统平台 z/OS 而言，REXX 以其与 z/OS 系统组件的广泛交互支持的特点、强大实用的文本处理功能以及极其简单灵活的结构化语法，成为大型机平台上使用最为流行的一种脚本语言。熟练掌握 REXX 的语言特性能极大提升大型机系统管理的效率和质量，也为各种进一步的复杂应用与扩展提供了可能性。本书作者在实际工作中发现，REXX 语言相关资料颇为繁杂，且由于其语言本身的灵活性，相关的应用描述和技术实践分散于不同的 IBM 官方发布的相关文档章节以及开发交流社区的文章之中，加之有中文详述的提及更是少之又少，造成了一些实践工作中学习的不便。本书作为第一本系统讲述大型机 REXX 编程实践的中文图书，旨在通过汇编总结相关文档中与日常应用最为紧密的知识点，并结合作者大量的实际应用经验和体会，力图使读者对 REXX 语言编程有一个直观全面的认识，为更加深入钻研 REXX 编程技术以及大型机系统管理的复杂扩展应用打下基础。

本书在编写、完稿至出版期间一直得到 IBM 公司大学合作部的大力支持，特别是李晶辉经理和万泽春经理对本书出版的支持和肯定；黄小平高级工程师对本书内容的编写、定稿都提出了宝贵的指导性意见；同济大学软件学院为本书的出版提供了有力的支持；软件学院的很多学生包括但不局限于郭意亮、张璇华、冯学奋、刘樽鹤和瞿苑等都参与了本书的文字校稿工作；本书还获得“IBM 大学合作项目书籍出版资助计划”的资助，并获得“同济大学十二五规划教材”支持，在此一并表示衷心的感谢。

本书遵循由浅入深的原则，采用了一般计算机编程语言讲解的通用逻辑顺序，大致可分为概括介绍、语法与程序规范、子系统交互以及执行/调试/实例分析四大部分，全书穿插了常用知识点与实例分析，希望对读者有所启迪。在成书过程中作者虽查阅了大量文献、编写调试实例程序验证力图严谨，但难免有所疏漏，望各位业内同仁以及读者予以斧正。

作　者

2012 年 2 月于同济大学

CONTENTS





第 1 章

主机脚本语言概述

脚本语言（Script Programming Language）是为了缩短传统的编写—编译—链接—运行（Edit-Compile-Link-Run）过程而设计的计算机编程语言。各种不同的计算机平台中存在很多可用的“组件”，人们开发了对应的系统脚本语言，使之能够快速组合这些系统平台中可用的“组件”，从而完成特定需求的脚本开发。

在设计一个应用程序时，用户可能需要考虑选择编译型语言或者解释型语言来编写程序源代码，而这两种类型的语言都有各自的优缺点。通常，确定使用解释型语言是缘于开发时间的限制或者未来程序的修改方便的考虑；而在确定使用解释型语言的同时也付出了代价：即系统需要以更高的执行开销换取了开发速度。这是由于在每次执行时，解释型语言的每一行源码需要被重新翻译解释，对系统而言意味着巨大的开销。因此，解释型语言通常更加适合于即兴需求，而不是那些预定需求。

脚本语言和一般程序设计语言的一个重要不同在于脚本语言通常是被解释执行，而系统程序设计语言通常是被编译执行。解释型语言由于不需要编译时间，而是通过快速的转换，使用户能快捷地编写并运行程序。由于这种特性，在系统管理以及测试中，程序员们编写了各种脚本来执行例行操作或者一次性任务。而脚本程序因为其灵活、简单、易于开发等特点，在主机系统管理领域有着广泛的应用。例如，系统管理员可能需要执行一些常规并且确定性的任务，例如输入相应命令、提交某个作业、检查数据集状态、为特定的程序分配数据集、打印文件等，通过编写相应的系统脚本，将这些操作所需的交互命令和任务组织成脚本文件的形式，能显著减少在这些例行任务上所花费的时间。将这些执行例行任务所需要的所有指令或操作集合到一个系统脚本程序中，可以节约大量人工操作时间、降低任务执行的错误概率、增强功能可复用性，并能极大提升系统管理工作的效率。z/OS 中所提及的系统脚本语言主要包括 CLIST、REXX 和 USS 中的 SHELL，后面将分别予以介绍。

1.1 CLIST 语言简介

CLIST 语言是一种解释型语言。和其他解释型语言编写的程序类似，CLIST 程序也易于编写和测试。术语 CLIST（读作 See List）表示“命令(Command)的序列(List)”，这是由于大多数的基本 CLIST 程序都是 TSO/E 的命令的序列。当用户调用一个这样的

CLIST 程序时，它将按顺序调用对应的 TSO/E 命令。

CLIST 编程语言主要用于以下几个方面。

- 执行日常的例行任务（例如输入 TSO/E 命令）。
- 调用其他的 CLIST 程序。
- 调用其他语言编写的应用程序。
- ISPF 应用程序（例如显示面板以及控制应用程序流）。

一个 CLIST 程序可以执行范围广泛的各种任务，主要可以归为以下三类。

- 执行日常例行任务的 CLIST。
- 作为结构化应用程序的 CLIST。
- 管理其他语言编写的应用程序的 CLIST。

1. 执行日常例行任务的 CLIST

用户可以编写相应的 CLIST 程序，它们能显著减少用户在这些日常的例行任务上所花费的时间。通过将执行任务所需要的所有指令集合到一个 CLIST 程序中，用户可以减少花费时间、键盘敲击次数以及任务执行中的错误，并且提升用户的工作效率。一个 CLIST 程序由单纯的 TSO/E 命令组成，或者由 TSO/E 命令与 CLIST 语句所混合组成。

2. 作为结构化应用程序的 CLIST

由于 CLIST 语言包括了编写一个完整的、结构化的应用程序的基本工具。任何 CLIST 都可以调用另外的 CLIST 程序，并作为一个“嵌套的”CLIST 程序引用。CLIST 同样包含了独立的子程序，称之为“子过程”。嵌套的 CLIST 程序和子过程使用户能够将 CLIST 程序划分成逻辑单元，并将公共功能在一处实现。

在交互式应用程序中，CLIST 可以调用命令显示 ISPF 面板。反之，ISPF 面板也可以根据用户在面板上的输入，来调用对应的 CLIST 程序。

3. 管理其他语言编写的应用程序的 CLIST

假设用户需要访问用其他语言编写的应用程序，但这些应用程序的相关接口可能不易使用或记忆，需要编写 CLIST 程序在用户和应用程序之间提供一个易于使用的接口，这要好过重新编写一个新的应用程序。一个 CLIST 程序可从终端上接收消息并通过程序逻辑判断用户需求。CLIST 程序可以设置环境并执行命令来调用程序，完成所相应的任务请求。

4. CLIST 语言的其他应用以及特色

除了请求执行 TSO/E 命令，CLIST 程序也可以执行更为复杂的编程任务。CLIST 语言包括了用户在开发大的结构化应用程序时所需用到的编程工具。CLIST 程序可以执行任何复杂度的任务，从显示一系列全屏面板到管理其他语言所编写的应用程序。

CLIST 语言特色如下。

- 具有处理数值数据所需的广泛的算术与逻辑操作符集。
- 具有很强的处理字符数据的字符串处理功能。
- CLIST 语句能很好地用于构建程序、执行 I/O 操作、定义和修改变量、进行错误

处理以及警示中断。

5. 如何执行 CLIST 程序

要执行一个 CLIST 程序，可使用 EXEC 命令。在 ISPF 的命令行中，命令开头需要输入 TSO。在 TSO/E EDIT 或者 TEST 模式下，在执行 EXEC 操作时使用 EXEC 子命令（执行在 EDIT 或者 TEST 模式下的 CLIST 程序只能够请求执行对应的 EDIT 或者 TEST 的子命令和 CLIST 语句，但用户可以使用 END 子命令以结束 EDIT 或者 TEST 模式，使 CLIST 程序能够请求执行 TSO/E 命令）。

1.2　REXX 语言简介

REXX（REstructured eXtended eXecutor）即重构的可扩充执行器语言，是 IBM 在 20 世纪 80 年代所发明的一种程序设计语言。主要用于 IBM 的主机系统，但在大部分其他平台上也可以找到对应解释器或编译器。REXX 是 IBM 随其大型主机、中型机操作系统和其他更低端操作系统一起捆绑的脚本和命令语言。REXX 几乎可以在世界上任何操作系统上运行。用户可以免费下载用于各版本的 Windows、Linux、UNIX、BSD、Mac OS 和 DOS 以及很多其他系统的 REXX。它甚至可以在用于手持设备的三大主流操作系统 Windows CE、Palm OS 和 Symbian/EPOC32 上运行。

REXX 也可用在 Java 环境中，例如 REXX 的一个分支叫做 NetRexx 能够与 Java 完全无缝的协同工作。NetREXX 程序可以直接使用任何 Java 类，并可以用来编写任何 Java 类。它将 Java 的安全性和性能特点带给了 REXX 程序，并将 REXX 的算术运算和简单性的特点带给了 Java。而作为单一语言的 NetREXX，可以同时用于脚本开发和应用程序开发。

1.2.1　一个简单的 REXX 程序

下面举例说明 REXX 语言的易学和强大。例如计算 1+2+…+10，然后把这些数字和所得的和显示出来。REXX 有强大的字符和数字处理功能，可以轻松实现。在这里也可以看出 REXX 程序很容易读懂。

```
/* REXX */
/* Count to ten and add the numbers up */
sum = 0
do count = 1 to 10
   say count
   sum = sum + count
end
say "The sum of these numbers is" sum"."
```

1.2.2 REXX 语言的一些不足

本章介绍 REXX 程序中存在的一些不足。

1. 难以维护

REXX 作为一种结构化语言，它使程序和算法能够以清晰和结构化的方式书写。但是，因为 REXX 的程序格式非常自由，变量的使用非常灵活，使得对不规范的 REXX 程序难以维护。所以在编写 REXX 程序时，应该尽量在开始部分写清楚程序的作用及主要的算法，在关键语句和变量处加以注释说明。

2. 执行效率不高

REXX 是一种解释型和编译型语言。正如前面提到的那样，这种解释型语言和其他诸如 COBOL 语言的不同之处在于，在它执行之前不需要编译 REXX 源程序。也正是因为 REXX 的这一特性，导致源程序在运行时才被逐句解释执行，效率比编译执行的语言要差。但是用户可以选择在执行之前编译 REXX 源程序生成可执行代码（Load Module），从而大大提高其执行速度。

1.2.3 REXX 语言的主要应用

REXX 编程语言的典型应用如下。

- 执行日常例行任务，例如输入 TSO/E 命令。
- 调用其他 REXX 程序。
- 调用其他语言编写的应用程序。
- ISPF 应用程序。
- 对问题的一次性快速解决方案。
- 系统编程。

一般而言，REXX 可以方便地对主机文件进行读写等操作。如果有大量的主机文件需要手工查找、修改等并且这些操作都具有共性，不妨使用 REXX 帮忙解决。例如，用户想将一个程序库中所有使用了某一些子程序的主程序全部找来;就可以用 REXX 解决;如果让用户对所有程序都加上一行一模一样的句子，这样手工修改起来势必会很麻烦，而且工作量也非常庞大，就可以考虑写一个 REXX 程序；如果用户想在一个程序中访问两个数据库，在 COBOL 中实现起来相对麻烦，而 REXX 就显得方便很多；还有 REXX 可以通过调用 SDSF 服务查找打印用户需要的作业信息，通过调用系统命令显示或修改系统配置等。后面将详细介绍关于 REXX 语言语法和应用的有关内容。

1.3 USS 的 Shell 简介

USS（UNIX System Service）所提供的 Shell 功能，顾名思义，就是 z/OS UNIX 交互接口所提供的面向 UNIX 命令和 Shell 语言的解释器。所支持的 Shell 脚本语言同标准

UNIX 的类似，z/OS UNIX 有两个版本的 Shell，即 z/OS Shell 和 tcsh Shell ,它们统称 z/OS UNIX Shell。

z/OS Shell 基于 UNIX System V Shell，其中部分功能源自著名的 UNIX Korn Shell。该 Shell 遵守 POSIX 标准 1003.2，并被采用为 ISO/IEC International Standard 9945-2: 1992。同时，z/OS Shell 向上兼容 Bourne Shell。

tcsh Shell 完全兼容并加强了 Berkeley UNIX C Shell，csh 是一个通用的语言解释器，既可用于交互式 Shell，也可用作 Shell 脚本的解释器。

Shell 脚本语法简要介绍如下。

- Shell 命令的组合
- While 循环
- For 循环
- DO…..DONE
- IF…ELIF…ELSE….fi
- TEST 用于条件测试，下例中代码用于目录的测试

```
if
    test - d $ 1
then
    echo "$ 1 is a directory "
fi
```

可以看到，Shell 脚本中可以使用 z/OS UNIX Shell 命令、分支语句、变量以及环境设定等。同一般的 UNIX Shell 脚本类似，USS 的 Shell 程序就是一个包含若干行 Shell 或 z/OS UNIX 命令，依据 Shell 的语法规则所写成的文件。解释器将 Shell 命令以脚本形式书写和存储，并以命令序列的方式在 Shell 下解释执行。如果用户熟悉其他平台下的 Shell 脚本书写，相信对 USS 下的 Shell 应该能够很快掌握运用。下面是一个 Shell 脚本的示例，该例中脚本程序检测某个 Java 类（java_memory.class）是否存在，如果存在，则执行该类；若不存在，脚本检测该类的源代码（java_memory.java）是否存在，如果存在，则将源码编译后执行。

```
if
  test -f  java_memory.class
then
  echo "the Java program is present. Let's execute it now"
  java  java_memory
elif
  test -f  java_memory.java
  then
    echo "the Java source is there . Let's compile it first!"
    javac java_memory.java
    echo "Now we will execute the Java code."
    java  java_memory
```

```
    else
      echo " there is no Java source present to compile"
fi
```

z/OS 的 Shell 编程环境以及所支持的 Shell 脚本，与 TSO/E 环境以及所支持的 CLIST 和 REXX 脚本类似，都是将完成特定任务或者一组任务所对应的命令或者所调用的相关过程，组织成脚本程序的形式，并由对应的环境解释执行，提高与环境交互工作的效率。

第 2 章

REXX 简介

REXX 可以用于多种平台之上，这是由 REXX 本身设计的特点所决定的。REXX 以约 20 条指令作为一个小的核心。这个核心周围有大约 70 个内置函数。这种设计使得 REXX 本身可以得到最大程度的扩展和延伸，使其可以使用包括数据库访问、GUI、XML、Web 服务器编程、MIDI 接口在内的各种形式的接口。这些易扩展性不仅使用户可以快速开发出简易程序，也可以使 REXX 用于持久健壮性程序的开发。事实上，借助特定的跨平台接口，REXX 脚本在 Linux、Windows、大型主机、手持设备之间具有广泛的可移植性。REXX 内核结构如图 2-1 所示。

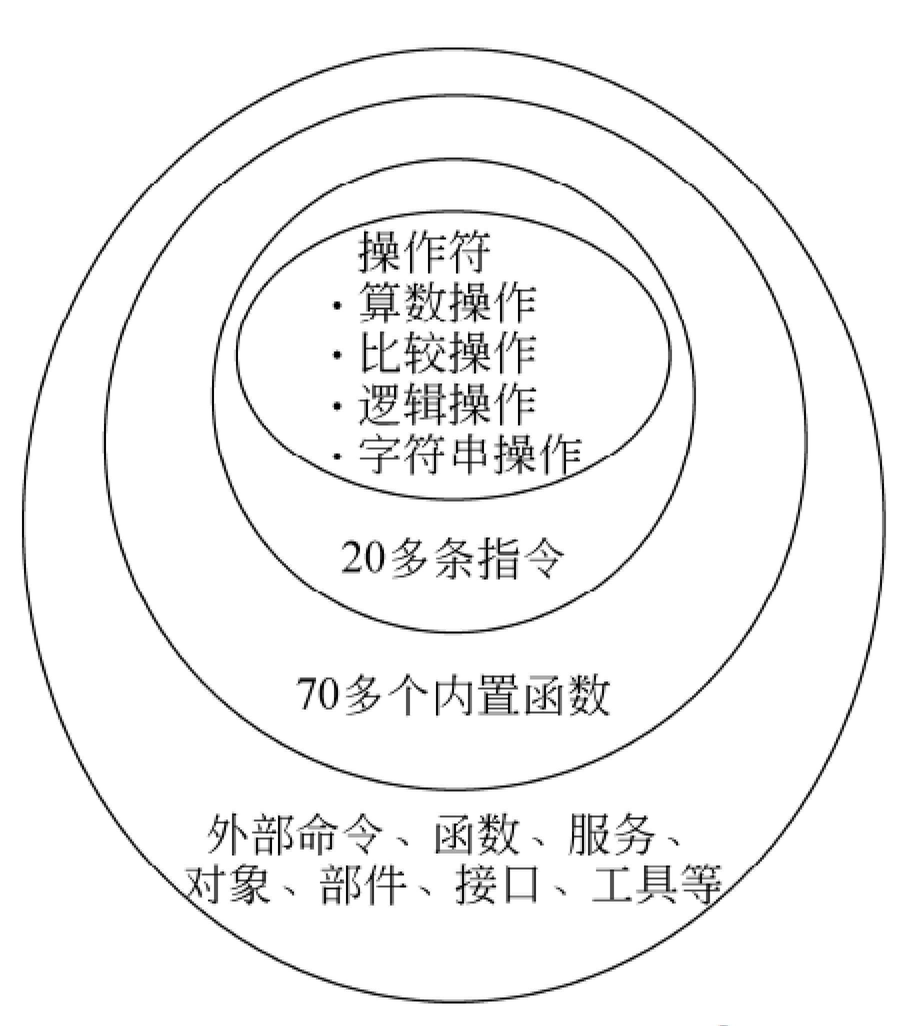


图 2-1　REXX 内核结构[①]

本章将详细介绍有关 REXX 语言本身的功能和使用，其中对语言语法规范的介绍是与具体平台实现无关的通用性内容。

2.1　REXX 的发展历史

REXX 最初创建于 1979 年，由当时 IBM 英国实验室的工程师 Mike Cowlishaw 设计，

① 摘自 http://www.ibm.com/developerworks/cn/db2/library/techarticles/dm-0508fosdick/index.html

作为一种新的脚本语言，其主要目的是为了替代 EXEC 和 EXEC2①语言。1982 年，REXX 被加入 VM/System 的第三个版本，正式作为 IBM 的产品推出。

REXX 被设计为一种可以运行于所有操作系统平台上的脚本语言，最初在主机平台上使用。从这个角度来看，REXX 可以说是 TCL 和 Python 等脚本语言的先行者。此外，REXX 设计者的另一目标是将 REXX 设计为 PL/I 脚本语言的简化版本，以便于程序员的使用和学习。

REXX 推出以后，多年来 IBM 致力于旗下各系列的操作系统都能够支持 REXX 的运行，包括 z/OS、VM/CMS、VM/GCS、MVS TSO/E、AS/400、VSE/ESA、AIX、CICS/ESA、PC DOS 以及 OS/2。同时，REXX 也被其他厂商的操作系统支持，包括 Novell NetWare、Windows、Java 以及 Linux②。

1996 年，美国国家标准协会(American National Standards Institution，ANSI)发布了 REXX 的标准 ANSI X3.274–1996。

20 世纪 90 年代中期，两种 REXX 的新版本逐渐开始流行，REXX 开始支持面向对象。

（1）NetRexx：REXX 的 Java 版本，将 REXX 编译/解释成 Java 二进制代码。NetRexx 完全没有保留字，并且使用了 Java 对象模型，因此并不是完全向上兼容于传统的 REXX。

（2）Object Rexx：作为 REXX 的面向对象版本，Object Rexx 基本完全向上兼容于传统的 REXX。

REXX 在开源的道路上也一直走在前面。早在 20 世纪 90 年代就有多个免费版本的 REXX 被广泛使用。2000 年前后，REXX 语言联盟（RexxLa.org）③取代 IBM，全面负责 REXX 语言的维护和管理工作。2004 年 10 月，IBM 宣布将 Object Rexx 的源代码开源；2005 年 2 月，Open Object Rexx（ooRexx）的第一个版本正式推出。

目前，REXX 主要运行在 Windows、Linux、AIX、z/OS、z/VM 等平台上。

2.2 各种版本的 REXX

由于历史原因，REXX 的大部分早期实现已不再是人们关注的焦点。然而，目前流行的几种版本的 REXX 实现依然可以用于包括 Linux、Windows 和 MacOSX 在内的多种操作系统平台，这些 REXX 版本包括 Regina、NetRexx、Open Object Rexx（ooRexx）。

---

① EXEC 和 EXEC2 都是运行在 IBM Virtual Machine/Conversational Monitor System（VM/CMS）操作系统上的解释型脚本语言，后被 REXX 所取代。

② REXX 的其他版本也运行在 Atari、AmigaOS、UNIX（many variants）、Solaris、DEC、Windows、Windows CE、Pocket PC、DOS、Palm OS、QNX、OS/2、Linux、BeOS、EPOC32、AtheOS、OpenVMS、OpenEdition、Apple Macintosh、Mac OS X 等平台上。

③ RexxLa.org 全称为 The Rexx Language Association，是独立的非营利性组织，致力于推广和使用 REXX 语言的应用。

1. Regina

Regina 是本地执行文件，用户可以从免费软件源代码的方式获得，或者直接获取诸多操作系统平台已经预编译的安装包，用户可以像安装任何其他语言解释器一样安装它。Regina 的两大目标是：

（1）100%支持 ANSI 标准。

（2）尽可能地支持更多平台。Regina 的 3.1 版本已经实现了第一个目标，目前 Regina 完全支持所有的 ANSI 特性，而 Regina 目前也已经支持了大多数主流操作系统平台①。

Regina 最新的版本是 3.5，可以访问其官方网站：http://regina-rexx.sourceforge.net 以获得更多信息。

2. NetRexx

IBM NetRexx 是 REXX 和 Java 的“混合物”，结合了 REXX 语法清晰易用的特点，Java 语言面向对象的特性，以及 Java 平台的健壮性和可移植性。因此，NetRexx 可以说是真正意义上的通用语言——适用于脚本开发和应用开发，同时既支持解释机制也支持编译机制。NetRexx 在语法上继承于传统的 Rexx，同时类似于 Jython 或者 Jacl，NetRexx 将 Rexx 源代码编译为 Java 字节代码，并可以在 JVM 中运行作为结果生成的 .class 文件。NetRexx 被编译成 Java 字节码后，可以运行在任何 JVM 上。（最初 NetRexx 仅支持 Java 1.0 版本的 JVM，现在已经不受限制。）

2011 年 6 月 8 日，IBM 宣布将 NetRexx 的 3.0 版本开源，并交给负责管理该语言的 Rexx 语言联盟管理。

3. Open Object Rexx

Open Object Rexx 作为 Object Rexx 开源后的免费版本实现，继承了 Object Rexx 的特性，包含了传统 Rexx 语法的以人为本的特点，以及目前更流行的面向对象特性（ooREXX 支持继承、多态、数据封装等特性）。用户可以按照传统 Rexx 的编程方法进行编程，或者按照面向对象的方式编程，甚至将这两种方法混合在一起。此外，ooRexx 的向上兼容性使得其依然可以被 IBM 早期的 REXX 编译器支持，同时其他开源版本的 REXX 编译器也能很好地支持 ooRexx。

Open Object Rexx 的最新版本是 4.10，可以访问其官方网站：http://www.oorexx.org 以获得更多信息。

2.3 主机上的 REXX

通常用户在主机平台上使用的 REXX，是 REXX 的 TSO/E 实现版本，TSO/E REXX 可以在 z/OS 下的任何地址空间中运行。在 TSO/E 环境下，用户编写的 REXX 脚本可以

① Regina 目前支持大部分 UNIX 平台（Linux、FreeBSD、Solaris、AIX、HP-UX 等），以及 OS/2、eCS、DOS、Win9x/Me/NT/2000/XP、Amiga、AROS、QNX4.x、QNX6.x、BeOS、MacOS X、EPOC32、AtheOS、OpenVMS、SkyOS 和 OpenEdition.

调用 TSO/E 服务并且在 TSO/E 的地址空间中运行。同时，用户也可以编写 REXX 应用并使其在 TSO/E 外的地址空间中运行。

此外，z/OS UNIX 对 TSO/E REXX 进行了一系列扩展，使得 REXX 程序拥有了访问 UNIX 可调用服务的能力。这一系列扩展被称为 syscall 命令，其命名与其对应的可调用服务一一对应。 关于 z/OS UNIX 扩展的详细信息，请参考 IBM 白皮书 *z/OS Using REXX and z/OS UNIX System Services*。

2.4 REXX 特性

REXX 语言的特性表现在以下几个方面。

1. 用途多样

REXX 既适合编程初学者使用，又适合经验丰富的专业从业人员，这样的特性在编程语言中并不常见。

这是因为 REXX 具有通用的语言结构、语法的高可读性（阅读 REXX 程序几乎和阅读普通的英语文章一样轻松）以及自由的书写规范（不同于 COBOL、REXX 几乎没有格式要求）等特点，这使得刚涉足编程领域的新手非常容易掌握这门语言。

同时，REXX 对于其宿主环境的命令良好支持（如在主机 TSO/E 环境下，用 REXX 脚本调用 TSO/E 命令非常容易），使得用 REXX 编写的应用能够支持更为强大的功能，同时也扩展了 REXX 本身的计算能力，因此有经验的编程人员常常要使用 REXX 来帮助完成系统的日常维护工作，或者根据需求编写一些实用有效的 REXX 应用。

2. 可读易用

REXX 语言使用了很多可读性强或者使用广泛的关键字和指令，例如 SAY、PULL、EXIT、IF-THEN、DO-END 等。不同于某些初等级别的编程语言使用缩写作为关键字，REXX 的关键字和指令使用常见的单词，这样大大增强了语句的可读性。同时，REXX 对标点、特殊字符以及符号的需要已经减少到最小。

REXX 的语法构建始终以良好的可读性和简易性为标准，这使得使用者在书写、调试、维护程序时，更为简单和方便。

3. 书写自由

相比于其他编程语言，REXX 对书写规范的限制极少，单条或多条的语句既可以单行书写，也可分成多行；语句不需要像 COBOL 或者汇编语言一样从特定的某列起始；在实际编程中，用户可以在一行语句中跳过多个空格，甚至完全跳过整个空行；REXX 对字母的大小写不敏感，用户可自由选择输入大小写或混合输入，其程序语义不受影响。此外，和典型的主机编程语言（COBOL、PL/I、主机汇编语言）的另一大不同就是，REXX 对行标号也无需求。

REXX 没有全局性的保留关键字，这使得用户不必去了解那些不需要用到的关键字，也对语言的应用扩展意义重大。

4. 单一数据类型

REXX 只有一种单一的数据类型，就是字符串，因此变量使用时不需要特别声明。REXX 通过变量的内容和使用情况动态地定义变量。在需要时，REXX 将自动转换数据，而其中的很多细节工作系统自动化完成，对用户而言是透明的。

例如，如果用户将“字面上”为数字的字符串赋给一个变量，那么用户就可以对该变量上实施相应的算术运算，而不用考虑其数据底层的含义和表示——这个特性确实非常吸引人——它使得程序员的编程工作变得非常方便和高效。

下面的例子是一个数据表示的代码片段。

```
x = 1               /* 变量 x 表示字符串"1" 同时也代表数字 1        */
y = "2"             /* 变量 y 表示字符串"2" 同时也代表数字 2        */
say  a + b          /* 显示 a+b 的结果为数字 3 (作为数字进行运算) */
say  a || b         /* 显示 a 和 b 的字符串拼接结果为"12"           */
c = "Hello"         /*变量 c 表示字符串"hello"                      */
say  a + c          /* 显示结果出错 无法完成类型转换                */
                    /* SYNTAX:  Bad arithmetic conversion (error 41) */
```

同时，REXX 对于变量的长度也并没有限制，在内存允许的情况下，REXX 变量的长度可以任意长。然而，REXX 对变量名的长度是有要求的，其上限为 250 个字符（实际上这是一个相当宽松的限制，因为几乎没有人将变量名定义到如此之长）。

REXX 对于变量名的上限如此之高，很大程度上是为了更好地支持复合变量（Compound Variable），复合变量的概念与数组非常接近，通过对于复合变量的使用，极大地增强了 REXX 的功能性和实用性。下面就是一个复合变量的例子。

```
NAME.Y.Z
```

关于复合变量的详细信息，详见第 3 章。

5. 强大的内置函数

REXX 提供了强大的内置函数，可以完成诸多的文本和数字的查找、比较和处理等工作，同时也提供了格式化操作、算术计算以及位操作、输入输出、转换等功能。

REXX 的字符串操作功能十分强大，并且由于语言本身对字符串的长度没有限制（包括其代表数字时），因而使得用户能够十分方便地实现对字符串拼接、分解、匹配等操作，完成各种文本处理和输出输入，从而免去了程序员在使用其他编程语言时所经常面对的字符串和输入输出处理等带来的困扰。

6. 跟踪调试方便

当 REXX 程序运行错误或异常时，解释器/编译器能给出对应的错误/异常的解释说明，并显示于运行终端。此外，使用 TRACE 语句也能够帮助用户方便快捷地调试跟踪程序。

REXX 本身包含了一个异常及错误处理程序，可对很多常见的错误条件（例如语法错误或算术上溢）设陷，并将控制流转移到专门的错误或异常处理例程中。异常设陷是

一种通过单独一个例程来管理常见错误的标准方式。

7. 编译解释型语言

TSO/E 将 REXX 作为一种解释型（interpreted）语言实现。也就是说，当一个 REXX 脚本运行时，语言解释器可以直接运行每一条 REXX 语句，而无须事先将程序编译（compile）为机器语言，也无须在运行之前先做链接（link）的工作。

同时，IBM 也支持使用 REXX 编译器，而事实上 IBM 针对 REXX/370 的编译器可以为开发者和使用者带来巨大的好处。这些好处包括：提升性能，降低系统装载量，保护源代码和程序，提升效率、质量以及编译后程序的可移植性。

2.5 REXX 的组成

REXX 拥有丰富多样的组件，这使得 REXX 成为编程者的一大有力工具。

REXX 主要包含以下几部分。

（1）**指令**：REXX 中包括 5 种类型的指令，分别是关键字、赋值、标签、空子句和命令调用。关于指令的详细内容，将在第 3 章中具体介绍。

（2）**内置函数**：此类函数由编译器直接支持，因而能够提供方便的处理功能。

（3）**TSO/E 外部函数**：此类函数由 TSO/E 环境支持，当 REXX 进行某些特定操作时，TSO/E 外部函数会与系统进行交互。

（4）**数据栈函数**（data stack）：数据栈提供数据的临时存储功能，以辅助 I/O 操作以及其他类型的操作。

2.6 第一个 REXX 程序

按照学习各编程语言的惯例，第一个 REXX 程序是一个 Hello World 小程序，其代码如下。

```
/* REXX */                          /* 标志出该脚本为 REXX 脚本 */
SAY "WHAT'S YOUR NAME?"             /* 输出一句提示语句 */
PULL NAME                           /* 将用户输入存入变量 NAME */
SAY 'HELLO WORLD! THIS IS' NAME     /* 输入结果 */
```

Hello World 小程序的主要功能是让用户输入自己的名字，然后把欢迎语句和用户的名字一起打印出来。其代码非常简单，总共只有 4 行。

第一行的 REXX 注释，是为了帮助主机系统区分 REXX 脚本和 CLIST 脚本。在默认的情况下，系统会优先将此类脚本作为 CLIST 脚本来解释。而如果在程序的首句标注了/* REXX */，则系统会将此脚本作为 REXX 脚本解释。因此，在通常情况下，在主机环境下编写的 REXX 程序，都会在开头添加/* REXX */注释。

第二行的输出语句，将一些信息输出到屏幕，以提示用户进行操作。SAY 关键字的

作用即为向输出流中写入一行数据，跟在 SAY 之后的表达式可以是一个变量；也可以像本程序一样，直接是一行字符串；甚至可以是变量和字符串的组合输出。如果在 SAY 之后没有跟随任何内容，则输出一个空的字符串。

第三行的输入语句，旨在将用户的输入存入一个变量(此处即为 NAME 变量)。PULL 关键字的作用是将外部数据队列中队列的头部读取到指定的字符串中，在 TSO/E 环境下，外部数据队列即为数据栈。当数据栈为空的时候，PULL 语句会从终端输入或者是输入流中（如使用 JCL 执行 REXX 程序，可以指定 SYSTSIN 作为输入流）读取数据。在本程序中，PULL 语句从用户在终端的输入读取数据。

实际上，PULL 指令相当于 PARSE UPPER PULL 指令的简写，也即是将读取的数据以大写的形式存入到指定的字符串变量中。

另外可以发现，程序在未事先声明 NAME 变量的情况下，直接使用了该变量。这反映了 REXX 语言的便捷好用的特点，但同时也会带来一些问题，如果总是直接使用变量而不事先定义，那程序的可读性和易维护性都会有所下降。因此，建议养成一个良好的编程习惯，对于在程序中常用到的变量，还是要统一进行事先声明。

第四行的输出语句，将之前获得的用户输入和另一些信息一并输出到终端，可以看到，SAY 关键字之后的输出非常自由，可以将变量和字符串结合在一起同时输出。

此外，为了便于在 TSO/E 环境下进行 REXX 脚本编程，可以在编译数据集时以 REXX 模式进行高亮提示。在命令栏中输入如下语句，即可打开 REXX 语句提示。

```
HI REXX
```

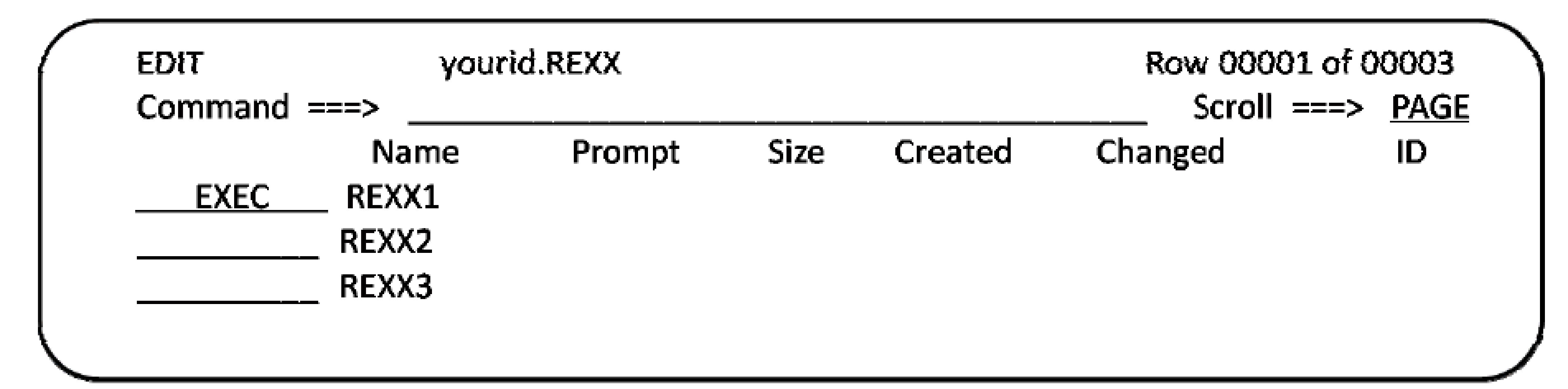
随后可以发现，程序代码（包括变量及操作语句）用绿色标示，关键字用红色标示，字符串（如 TSO/E 命令）用白色标示，而注释语句则用青色标示。这些高亮提示可以很好地辅助用户进行编程，以及帮助检查基本的语法拼写，减少错误的出现。

Hello World 程序的执行结果如下，到这里，第一个 REXX 程序的编写已经完成了，REXX 学习之旅迈出了第一步。

```
WHAT'S YOUR NAME?
Emma
HELLO WORLD! THIS IS EMMA
```

2.7 REXX 执行

执行 REXX 最简单的方式，是在进入存放 REXX 脚本的 PDS 数据集后，直接在需要执行成员左侧的命令栏中输入 EXEC，即可执行该程序。通过这种方式执行 REXX 程序的示例如图 2-2 所示。



```
EDIT              yourid.REXX                          Row 00001 of 00003
Command ===> ________________________________________   Scroll ===> PAGE
             Name        Prompt    Size   Created    Changed         ID
___EXEC___  REXX1
__________  REXX2
__________  REXX3
```

图 2-2　REXX 程序执行方式 1

当然也可以在命令栏输入完整的 EXEC 命令，显式调用一个 REXX 脚本。其结果如图 2-3 所示。

```
EDIT              yourid.REXX                          Row 00001 of 00003
Command ===> TSO EXEC 'yourid.REXX(REXX1)' exec______   Scroll ===> PAGE
             Name        Prompt    Size   Created    Changed         ID
__________  REXX1
__________  REXX2
__________  REXX3
```

图 2-3　REXX 程序执行方式 2

要注意的是，EXEC 命令最后的 exec 参数，是系统用于判断执行的脚本是否是 REXX 脚本的标志。

这种执行方式，适合学习编写 REXX 程序时进行练习操作时使用；然而对于日常使用的系统脚本，还是建议将编写好的 REXX 脚本放入对应的系统库（SYSEXEC 或 SYSPROC）中，以便于维护和使用。

如果 REXX 程序已经存放在了系统库中，则用户可以直接在命令栏指定执行该程序，而无须给出其所在的数据集名，图 2-4 就是这样一个示例。

```
--------------------------------------------------------------------------------
                          ISPF Primary Option Menu
Option ===>  TSO SYSLIB

0  Settings      Terminal and user parameters           User ID . : IBMUSER
1  View          Display source data or listings        Time. . . : 08:45
2  Edit          Create or change source data           Terminal. : 3278
3  Utilities     Perform utility functions              Screen. . : 1
4  Foreground    Interactive language processing        Language. : ENGLISH
5  Batch         Submit job for language processing     Appl ID . : ISR
6  Command       Enter TSO or Workstation commands      TSO logon : DBPROC9G
7  Dialog Test   Perform dialog testing                 TSO prefix: IBM100S
8  LM Facility   Library administrator functions        System ID : S0W1
9  IBM Products  IBM program development products       MVS acct. : FB3
10 SCLM          SW Configuration Library Manager       Release . : ISPF 6.1
11 Workplace     ISPF Object/Action Workplace
```

图 2-4　REXX 程序执行方式 3

除了前台调用的方式，还可以通过批处理（Batch）的方式，使用 JCL 提交作业来执行 REXX 程序。本例中执行了 yourid.REXX (REXX1)的 REXX 脚本，其 JCL 语句如下。

```
//EXECREXX JOB NOTIFY=&SYSUID
//TMP      EXEC PGM=IKJEFT01
//SYSEXEC  DD DSN=yourid.REXX,DISP=SHR
//SYSTSPRT DD SYSOUT=A
//SYSTSIN  DD *
  REXX1
/*
```

该程序的执行结果，可以查看 SDSF 中相应作业执行情况的信息，如图 2-5 所示。

```
SDSF OUTPUT DISPLAY     EXECREXX JOB06152     DSID   102    LINE 0          COLUMNS 02- 81
COMMAND INPUT ===>                                                          SCROLL ===> PAGE
******************************** TOP OF DATA **********************************
READY
 REXX1
THIS IS A REXX PROGRAM
READY
END
******************************** BOTTOM OF DATA ********************************
```

图 2-5 REXX 程序执行方式 4

2.8 REXX 调试

REXX 提供了方便的调试跟踪机制，方便用户对程序进行调试和问题诊断，主要的调试方法包括以下几方面的内容。

- TRACE 指令调试；
- RC/SIGL 特殊变量；
- 交互式调试跟踪机制。

1. TRACE 指令调试

TRACE 调试指令有两个常用选项，即 C（Command）和 E（Error)选项，在 REXX 程序使用 TRACE C 指令后，所有之后的命令调用将被追踪报告，在这些命令执行完成后，执行返回码亦会显示在屏幕上；若在 REXX 程序中使用 TRACE E 命令，则任何执行后产生非零返回码的命令都将被追踪报告，并将该命令的返回码显示到屏幕上。更多有关 TRACE 指令的有关内容，可参考 IBM 白皮书 *z/OS TSO REXX Reference*。

2. REXX 特殊变量 RC 和 SIGL

REXX 包含了两个特殊变量 RC 和 SIGL，这两个变量的值由系统赋予，可在任何需要的时刻用于表达式中；同时也可同一般变量一样，使用相关的 REXX 指令进行操作。

RC 表示返回码（Return Code）,其值在每次命令执行后，都将被重置为最新命令执行完成的返回码。在命令执行无错时，RC 置为 0；当有命令执行出错时，RC 将被置为出错命令的返回码。

SIGL 表示在 REXX 程序执行过程中控制流发生转变时的代码行号。在 REXX 程序

中使用函数调用、Signal 以及 Call 指令等方式都将使当前的程序控制流发生转向，此时系统将 SIGL 的值置为最近一次控制流转向发生时的代码行标号。

在 REXX 程序中，综合使用上述的特殊变量，以及 SIGNAL ON ERROR 指令可以帮助程序员定位错误发生的位置，方便进一步的程序调试，见下例：

```
/* REXX */
SIGNAL ON ERROR
"ALLOC DA(new.data) LIKE(old.data)"
...
"LISTDS ?"
...
EXIT

ERROR:
SAY 'The return code from the command on line' SIGL 'is' RC
/*显示：The return code from the command on line 8 is 12 */
```

3. 交互式调试机制

交互式调试机制使程序员可以交互式控制跟踪每一步的 REXX 程序执行。在需要的时候程序员可以暂停 REXX 程序的执行，并可选择继续跟踪下一条指令执行、插入新指令语句、重新执行前面的指令等操作，也可以改变或终止继续交互跟踪调试。通过该机制可使程序员交互灵活地控制 REXX 程序，方便地进行程序调试。可通过前面提到的 TRACE 指令的“?”选项或者 TSO/E 下的 EXECUTIL TS 命令选项启动交互式调试机制。

下例使用了 TRACE ?R (TRACE ?Results)指令可以交互调试 REXX 程序中的每条指令：

```
/***********REXX******************/
TRACE ?R
ARG dsname .
LISTDS dsname
IF RC = 0 THEN
SAY 'LISTDS is executed successful.'
ELSE
SAY 'Return code from LISTDS was' RC
```

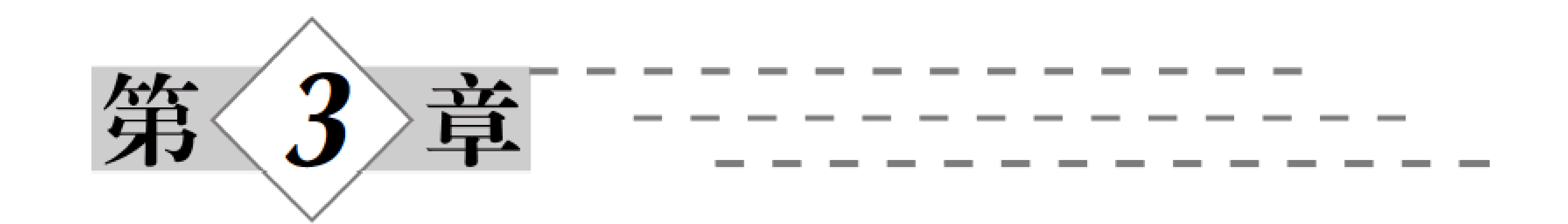
如果在使用 EXEC 调用该程序执行时，通过命令行传入给 LISTDS 的参数是 NONE（数据集 USERID.NONE 不存在），则会出现以下交互式调试结果：

```
3 *-* ARG dsname
  >>>   "NONE "
IRX0100I +++ Interactive trace.  TRACE OFF to end debug, ENTER to continue.
+++
```

按回车键继续下一步调试，显示出相应的错误信息和返回码如下：

```
        4 *-* LISTDS dsname
   >>>   "LISTDS HEHE"
IKJ58503I DATA SET SP005.NONE NOT IN CATALOG
   +++ RC(20) +++
```

由交互调试界面可知，TRACE OFF 可结束该调试画面。命令 EXECUTIL TS(Trace Start)/TE(Trace End)与 TRACE ?R/OFF 作用基本等效。更多有关命令 TRACE 以及 EXECUTIL 命令的有关内容，请参阅 IBM 白皮书 *z/OS TSO REXX Reference*。



REXX 语法

由于 REXX 有简单易学的语言特点以及自由灵活的编程风格，使得 REXX 编程并不复杂，相信接触过其他编程语言的读者可以很快写出一个简单的 REXX 程序，例如：

```
/*****************REXX******************/
SAY 'HELLO WORLD.'
```

基本的 REXX 组成包括了指令和内置函数两部分，此外 TSO/E 环境下的 REXX 还提供了外部函数以及数据堆栈函数等。本章着重介绍 REXX 的基本语法和指令的编写。

3.1 指令概览

有许多编程语言都对指令的编写形式有着严格的规定，如在 CLIST 中，指令必须为大写字母；在汇编指令中，要求从指定位置开始书写。但是在 REXX 中，对语法的要求比较宽泛，指令可以从任意位置开始写，在任意位置结束，一条指令可以跨越多行，一行也可以容纳多条指令，并且指令的大小写形式也是任意的，这就使得 REXX 编程少了许多语法上的约束，更加简单实用。

3.1.1 指令的语法规则

REXX 指令可以为大写字母、小写字母或者两者的混合。但是，在编译执行过程中，指令会被自动转化为大写形式，用引号引用的部分除外。

关于引号的使用，可以使用单引号或者双引号，只要相同的引号成对应用即可。如果使用 SAY 指令时忽略了引号，系统会自动将内容转换成大写输出，例如：

```
SAY This is a string.    /* 输入 */
THIS IS A STRING.        /* 输出 */
```

如果字符串中含有单撇号（'），也可以用双引号将其括起来，系统会自动识别相应符号的含义，并解析输出，例如：

```
SAY "This isn't a good idea." /* 输出 This isn't a good idea. */
```

3.1.2　指令的格式

REXX 语言的指令格式十分自由，一条指令可以从任意列写起，系统会忽略掉多余的空格和空行，如果一条指令在一行之内不能全部容纳，可以续行，此时在上一行的结尾加上逗号，表示续行，这种续行方式会在两部分指令之间加一个空格，例如：

```
SAY 'THIS IS AN EXTENDED',
 'INSTRUCTION.'

/* 输出 "THIS IS AN EXTENDED INSTRUCTION." */
```

如果不想加入空格，可以用（||）来进行续行，例如：

```
SAY 'THIS IS A STRING USING CONCA'||,
 'TENATION OPERAND.'

/* 输出 "THIS IS A STIRNG USING CONCATENATION OPERAND." */
```

一条指令结束用分号表示，也可用分号分隔一行中的几条不同的指令。如果一行只有一条指令，也可以不标记结尾，因为默认情况下一行的结束标识着一条指令的结束，例如：

```
SAY 'hello'; SAY 'HELLO WORLD'; SAY'GO ON'
指令的输出为:
hello
HELLO WORLD
GO ON
```

注释写在一对/*与*/之间，可以写在任意位置，也可以嵌套在命令中，最大长度为 250 个字节，可以跨行，建议使用规范统一的注释风格。

3.1.3　指令类型

在 REXX 中，有 5 种类型的指令：关键字、赋值、标签、空子句和命令调用。

1. 关键字

关键字指令通常起始于一个 REXX 关键字，它告诉语言处理器做哪些事情，例如，Say "Hello World!"指令中，关键字 Say 表示将字符串“Hello World!”显示到屏幕上；IF、THEN、ELSE 为三个关键字，共同构成了一个控制流指令，其中每个单独的关键字及其指示的内容形成一个子句（clause），它是指令的一部分。常见的关键字还有 EXIT、DO/END 等。

2. 赋值

即对变量赋值或者改变当前变量值。赋值操作计算语句右边的表达式，并将计算结果值赋给左边变量。例如 Variable = 6 ; Variable = Variable +3 等。

3. 标签

指后面紧跟冒号的名称符，出现在指令开始处。标签名称可以包含单、双字符，同一个标签可以出现在程序中的多处，但是在搜寻目标标签时，只执行最近的标签程序片段。例如“Program1 :”、“100.3:”等都是合法有效的标签。标签主要用于标识程序片断，经常用于子例程或函数之中。标签可用作某些指示性指令（如 SIGNAL、TRACE）的作用目标。

4. 空子句

空子句通常指注释或空行，其作用是增加程序的可读性。在执行过程中会被语言处理器跳过忽略。和其他编程语言类似，适当使用注释可以提高可读性，使程序更加便于理解和维护。特别地，为了在主机环境下区分 CLIST 程序与 REXX 程序（exec），通常在 REXX 程序的程序起始处，加上一行包含了 REXX 字样的注释。通过这样的首行注释可使系统将其识别为 REXX 程序进行处理，从而与 CLIST 区别。空行可用于适当的程序格式划分，同样是为了增强程序代码的可读性。

5. 命令调用（包括 TSO/E REXX 命令以及宿主命令）

当程序指令不是关键字、赋值、标签或者空子句时，将作为命令置于前面所定义的环境中进行处理。以下程序中出现的 TIME 将被作为 TSO/E 命令进行处理（这里特指在 TSO 地址空间下）。

```
/* TSO system */
TIME          /* 命令内容 */
```

3.2 变量和表达式

本节介绍 REXX 语言中变量和表达式的使用方法，变量和常量等组成了表达式，表达式又可以组成指令和函数，以及可执行的 REXX 程序。通过变量和表达式的运算可以实现想要得到的功能，是 REXX 编程的基础。

3.2.1 变量的使用

REXX 中变量的概念和其他编程语言类似，即代表程序中某值的符号名，变量可用于在不同时刻指代不同值，或者指代编程时刻所无法得到的值，例如程序内部变量或者外部文件名。前者可通过赋值语句赋予变量新的值（或初值），后者可通过相应的输入输出操作得到对应值。

1. 变量名称

变量名称可以包括：大小写英文字母以及@、#、$、?、!、.、_等特殊符号，还可以有 X'41'、X'FE' 这类双字符（Double-Byte Character Set，DBCS）。如果要使用双字符，需要在程序中指定 ETMODE。例如：

```
OPTIONS'ETMODE'        /*通过设置 ETMODE 引入双字符变量*/
<.S.Y.M.D>=10          /*DBCS 变量要用<>括起来*/
```

变量名称还有以下约束：第一个字符不是 0～9，也不能是圆点（.），长度不能超过 250 个字节。对于 DBCS 字符，每个字符占两位，SO 和 SI 各占一位。特别地，变量名字不能取 RC、SIGL 或者是 RESULT 等在 REXX 中有特殊意义的保留字。下面给出几个可行的变量名供参考 HELLO、A、@hi、go_、home、list4。

2. 变量取值

变量的值可以是数值常量，如整型、浮点型、带符号数等，也可以是字符串，其他变量的值或者表达式。如果变量没有初始化，那么它的值为变量名的大写字母表示。可能得到的变量值如下。

```
Variable = 12
Variable = 'this is a string'
Variable = varialbe2
Variable = 2 * 3 + 4
```

变量的赋值可以用 PARSE 指令、VALUE 内置函数、赋值表达式以及其他编程语言来实现。

3. 变量类型

REXX 变量包含三种类型：简单变量（Simple Variable）、复合变量（Compound Variable）和复合词干（Compound Stem）。

4. 简单变量

简单变量是指符号名称中不含任何句点（.）的变量，如 user_name,如果该变量没有对其显式赋值，则它所代表的值即为符号名称本身的大写，即为 USER_NAME。注意，REXX 对变量名称本身的大小写是不敏感的，所以不能用大小写来区分名字相同的两个变量。

5. 复合变量

复合变量是符号名称中含有一个或多个句点（.）的变量，由词干和词尾组成，分别遵循简单变量的命名规则，如 A.3、A.X.Y、B..1.2.等。变量的总长度（包括词干和词尾）不超过 255 个字符。复合变量的取值为词干的大写值加上词尾的变量值，如果词尾变量未赋值，就是对应的变量名称的大写值，词尾变量中可以有空值，如 TIME = 3，REXX.TIME 的值为 REXX.3。

复合变量可将变量值以分组的形式存放和访问。事实上，复合变量对应了其他编程

语言的数组（array）或链表（list）。在 REXX 中，“数组”的下标值不一定为数字，可以是任何有意义的字符串，便于开发和使用。

6. 复合词干

复合词干是复合变量中的第一个变量名称和第一句点所组成的。通常用于将复合变量所表示的整个变量集合赋予相同的初值，例如：

```
room. = "empty"
room.mine = "full"
say room.1 room.yours room.mine  /* 输出 "empty empty full" */
```

由此可以看出，每个复合变量的开头（首段变量加“.”）都是一个复合词干。

需要注意的是，复合变量的首段变量名通常是表示符号名称本身，而剩余部分的变量接受前面所赋予的值，例如：

```
Firstname= 'Jack'
Lastname = 'Green'
Student = firstname.lastname

SAY Student
/* 输出 " FIRSTNAME.Green" */
SAY Student.firstname.middle.lastname
/* 输出 " STUDENT.Jack.MIDDLE.Green" */
```

上例中，firstname.lastname 作为变量使用时，其首段变量（firstname）表示符号名称本身，作为词尾时，其值为被赋予的字符串值。

```
say Grade.Row.Col       /* 输出 " GRADE.ROW.COL" */
Row=8
Col=6
say Grade.Row.Col       /* 输出 " GRADE.8.6" */
Grade. = "Pass"         /* 注意 Grade 后面的"." */
say Grade.Row.Col       /* 输出 "Pass" */
Grade.8.6 = 'Fail'
say Grade.Row.Col       /* 输出 "Fail" */
```

注意区分上述两例中复合变量第一变量名 Student 和复合词干 Grade.的不同表示和意义。

3.2.2 表达式的使用

表达式是需要运算的语句，由数字、变量、字符串和操作符组成。操作符决定了运算的类型，有算术运算、比较运算、逻辑运算和串联运算。

1. 常量表达式

在 REXX 中表达式的使用和其他编程语言类似，如 3、1234.56、–1、1.23e+7、4.56E-3

等，可以用于表示数字常量。在算术运算处理中，可通过使用 NUMERIC DIGITS 指令指定数值运算的有效位数，例如，在 NUMERIC DIGITS 指令默认设置（9 位）下，12345678910111213l4 在运算处理时，被作为 1.23456789E+18 进行处理。对字符串常量，在不至于混淆的情况下，单引号和双引号皆可使用，即'literal string'和"literal string"均可使用，在引号内的字符串中，可使用任何符号，包括同类型引号本身。

2. 算术表达式

算术表达式使用到的运算符包括：+（加）、–（减）、*（乘）、/（除）、%（取商）、//（取余）、**（乘方） 以及正负前缀+/–。

注意除法运算中，两个整数相除得到的结果可以是浮点数。如 7/2，结果是 3.5。

按照从左到右的运算顺序，优先级为先计算括号中的，然后按照运算符的优先级依次计算。具体优先级情况见后续表格。

3. 比较表达式

比较表达式通过比较运算返回真或假，用 1 和 0 表示。比较运算通常和 IF-THEN-ELSE 配合使用，用于判断程序执行的路径。比较运算的对象可以是数值和字符串。表 3-1 列出了一些常用的比较运算符。

表 3-1　比较运算符

运 算 符	逻辑判断	运 算 符	逻辑判断
=	等于	>=、¬<、\<	大于等于；不小于
¬=、\=、><、<>	不等于	<=、¬>、\>	小于等于；不大于
>	大于	==	严格等于
<	小于	¬==、\==	严格不等

注意，REXX 中的非符号，可以用¬或\表示，两者作用相同。严格比较（不）等于，表明将表达式两端进行完全比较，包括了空格和大小写。而（不）等于运算只是比较两端字符串的内容是否相同，如下例。

```
'HELLO' = hello         返回 1
'hello' /== hello       返回 1
'hello' == hello        返回 0
```

4. 逻辑表达式

逻辑运算与比较运算的返回值相同，结果为真时，返回 1，反之返回 0。常用的逻辑运算符有：&（与）、|（或）、\（非）、&&（异或），遵循一定的优先级规则，为了避免混淆，可以用括号来标识运算顺序。

逻辑操作符号组成的逻辑表达式通常用在 IF-THEN、DO-WHILE、WHEN-THEN 等关键字指令构成的条件子句中，用于控制分支流或者选择屏蔽某些条件，例如：

```
IF((A < B) | (J < D)) & ((M = Q) | (M = D)) THEN ...
```

5. 串联表达式

串联操作将指令中的两项连成一项，可以是字符串、变量、表达式和常量，串联操作常用于格式化输出。常见的串联操作符如下。

（1）空格，连接两端的项并在其中加入一个或多个空格分隔，两项之间默认设置为一个空格连接。

```
Say X      Y z     /* 结果显示为"X YZ" */
```

（2）“||”，直接连接两端的项，中间不插入空格。

```
8||(3 * 3)         /* 结果显示为 89 */
```

（3）直接相邻，连接两个直接相邻不同项，中间不加入空格。

如变量 percent 为 60，则 percent'%'表示 60%。

以上运算符的优先级从高到低如表 3-2 所示。

表 3-2　运算符优先级（从高到低）

运算符	符　号	运算符	符　号
前缀操作符	\、–、+	串联操作	空格、‖、直接相邻
乘方（指数）	**	比较操作	==、=、>< 等
乘除	*、/、%、//	逻辑与	&
加减	+–	或、异或	|、&&

3.3　关键字指令

关键字指令是以关键字开始的指令，可用于改变控制流，为编程人员提供系统服务等，有些关键字指令，如 DO，可以包含嵌套的指令。凡是以关键字开始的指令都不是系统命令，如 ARG(BREAD) REST 是一个 ARG 关键字指令，而不是系统的 ARG 内置函数调用。

在 DO、IF、SELECT 指令中，关键字必须成对出现，否则系统会提示语法错误。在其他情况下，关键字不是保留字，可以用于标签或者变量名，但这种做法是不推荐的。还有一些子关键字，它们对应于特定的关键字指令，如 VALUE 和 WITH 分别是 ADDRESS 和 PARSE 的子关键字。

下面介绍一些常用的关键字及其对应的指令，如需了解更多关键字的使用方法，请查阅 IBM 白皮书 *TSO/E REXX Reference*。

3.3.1　ADDRESS 关键字

REXX 中可以使用命令,它们是发往外部环境的字符串，REXX 可通过显式地使用 ADDRESS 指令来指定命令的目标执行环境。它常用的表示方法有以下两种。

1. ADDRESS environment （expression）

语句中，environment 为一个常量字符串，指定环境名称，是可执行 REXX 命令的外部程序或进程的名字。expression 为要执行的命令，通过一定形式的转换发送到 environment 所指定的目标环境，该命令执行后环境变量会恢复到之前的值，此时 RC 会被赋值，返回命令执行的结果状态，RC 为“0”则表示命令执行成功。

```
ADDRESS LINKMVS "program p1 p2"
/* LINKMVX 为宿主环境,提供链接功能,该语句将链接到可执行模块 program 并提供参数 p1
p2 */

ADDRESS MVS "MAKEBUF"        /* 在 MVS 下执行 MAKEBUF 命令 */
ADDRESS TSO "DELSTACK"       /* 在 TSO 环境下执行 DELSTACK 命令 */
```

如果未指定 expression，只是定义了 environment 变量的值，则执行环境的更改是永久性的，在此之后的所有命令都会发送到相应环境中执行，直到下一次 ADDRESS 命令显式地使用。

```
ADDRESS MVS                  /* 此后的命令都在 MVS 环境下执行 */
```

2. ADDRESS （VALUE） expression

当要执行的环境名称需要通过运算得出时，可以使用 VALUE 关键字，执行结果等同于第一种方法。其中，expression 表示目的环境的名字，通常是一个表达式。如果 expression 以特殊字符开头，如括号等，则 VALUE 可省略。

```
ADDRESS ('ENVIRONMENT'||NUM)
/*上句等同于 ADDRESS VALUE 'ENVIRONMENT'||NUM, 首先通过运算得到
'ENVIRONMENT'||NUM 所代表的环境名称,然后再设定此后的命令都在对应的环境中执行 */
```

REXX 提供了许多宿主命令执行环境，可以用 ADDRESS 指令来转换，这些环境提供了使用 TSO/E、MVS、ISPF 不同功能和服务的方法。

如果使用了 CALL 命令，当前的环境值会被保存起来，当函数调用返回时再读出之前保存的信息，这些环境信息也可以通过 ADDRESS 内置函数得到。

3.3.2　ARG 关键字

AGR 关键字用于读取用户提供的参数值，并向函数或子例程中对应的变量赋值，它的指令格式如下。

```
PARSE UPPER ARG (template_list)
```

其中 template_list 是一个或一组变量串（template），用逗号隔开，每一个 template 由用空格或括号分隔的变量组成。

对于传递给函数或子例程的参数，编译器会先将字符串转换成大写值，然后再进行赋值等处理，如果想保留原格式，可以使用 PARSE ARG 指令。对于同一个字符串，可

以反复使用 ARG 或 PARSE ARG 指令，不会改变原字符串的内容。可处理的字符串的长度仅由系统参数决定。

```
/* 传递的字符串是"hello world" */
(PARSE) ARG X Y.          /* 执行结果是 X = hello Y = world */
PARSE UPPER ARG X Y.      /* 执行结果是 X = HELLO Y = WORLD */
```

如果要向程序传递多个字符串，可用逗号隔开，使得字符串序列中的每一个字符串依次接收数据。

```
/* 调用函数 SAMPLE('hello world',2) */
SAMPLE:
(PARSE) ARG string, num
/* 执行结果是 string = 'hello world', num = 2 */
```

3.3.3 SAY 关键字

SAY 关键字用于向终端输出一行字符，可以设置输出的位置。如 REXX 程序在 TSO/E 前端运行时，会输出到显示终端；在 TSO/E 后台运行时，会输出到系统输出流，即 SYSTSPRT；在非 TSO/E 环境下运行时，会输出到由 OUTDD 指定的位置。输出字符的格式会根据当前运行环境的设置来调节。SAY 关键字的使用在前面的例子中已经出现过很多次，需要注意引号和一些连接符号的使用方法。

```
NAME = REXX
SAY 'MY NAME IS ' NAME
/* 输出：MY NAME IS REXX */
```

3.3.4 PROCEDURE 关键字

PROCEDURE 关键字使用在子例程中，用于隔离子例程和主程序中变量的使用。当子例程返回时，恢复调用前主程序变量的值，并且在子例程中使用的变量都被舍弃，使用格式如下。

```
PROCEDURE (EXPOSE) (name)
```

PROCEDURE 关键字需是在子例程标签后的第一条指令。

```
/* REXX 主程序 */
X = 1; Y.1 = 'a'
CALL SUBPROC
SAY X K M           /* 输出 "1 K M" */
EXIT

/* 子例程 */
SUBPROC: PROCEDURE
```

```
SAY X K Y.1        /* 输出 "X K Y.1" */
K = 7; M = 3       /* K 和 M 的值不会返回到主程序中 */
RETURN
```

如果想和主程序共享一些变量，可以使用 EXPOSE 选项，例如：

```
/* REXX 主程序 */
J = 1; K = 6; M = 9 A = 'J K M'
CALL SUBPROC
EXIT

/* 子例程 */
SUBPROC: PROCEDURE EXPOSE (A) /* 共享 A、J、K 和 M */
SAY A J K M /* 输出 "J K M 1 6 9" */
RETURN
```

对于 EXPOSE 后面的变量，如果是简单变量，直接写出变量名字即可，多个变量用空格隔开，如果使用括号，则变量会被展开成对应的变量列表，如上例中，变量 A 以及它所代表的变量列表 J K M 都会被共享。

变量按照指定的顺序被共享，如下例，如果变量 Z.J 列在变量 J 之前，则对于 Z.J 来说，J 的值还未被共享，所以不是 1，那么就不能共享主程序中的变量 Z.1 了。

```
/* REXX 主程序 */
J = 1; Z.1 = 'A'
CALL SUBPROC
SAY J K M          /* 输出 "1 7 M" */
EXIT

/* 子例程 */
SUBPROC: PROCEDURE EXPOSE J K Z.J
SAY J K Z.J      /* 输出 "1 K A" */
K = 7; M = 3
RETURN
```

3.3.5　CALL 关键字

CALL 关键字用于调用函数和跟踪特定的事件。

1. 特定情况的跟踪

通过 CALL ON 和 CALL OFF 可以捕捉到特定事件或状态的发生，并且改变程序的执行顺序。用法是：CALL ON condition (NAME trapname)和 CALL OFF condition。

其中 condition 指定了要跟踪的内容，trapname 指定了跟踪后的处理程序，可以是内部例程，系统内置函数或外部例程，如果未指定 trapname，则默认跳转到 condition 指定的例程或标签处。

常见的跟踪内容有 ERROR、FAILURE、HALT、NOVALUE 和 SYNTAX，关于 CALL 指令对系统各事件的跟踪，在第 9 章有详细说明。

2. 调用函数或子例程

CALL function_name，通过指定例程的名字来调用，可以是内部例程、内置函数或外部函数，这些函数等的名称都是大写的。如果名字是用引号括起来的字符串，则编译器会认为是内置函数或外部例程。注意内置函数名称在引号内时要大写，因为在传递时不对引号中的内容进行大写转换。CALL 指令可以有返回值，存储在特殊变量 RESULT 中，而不是 RC 中。如果作为函数调用而未定义返回值，系统会报错。

当使用 CALL 指令时，系统搜索对应例程的顺序是，先搜索同一个 REXX 程序内部的例程，然后是系统的内置例程，最后是外部程序中的例程，所以要注意避免因名字相同而调用错误的情况。对于内部例程的调用可以嵌套，同时也可以递归调用自身。具体关于函数调用的方法请参见第 4 章。

3.3.6 DROP 关键字

DROP 可以使变量恢复到未初始化的状态，也就是使得变量的取值等于它们名字的大写值。使用方法：DROP （name），其中 name 是变量的名字。如果不加括号，表示要 DROP 对应名字的变量；如果加括号，表示要 DROP 由 name 定义的一组变量。可以一次 DROP 多个变量，之间用逗号或空格分开，按照顺序依次处理。

如果指定了未定义的变量或者同一个变量指定多次，不会产生错误信息。如果 DROP 的是一个复合词干，如 STEM.，那么所有以这个词干为开头的复合变量都将恢复初始值。

```
DROP A B.C       /* DROP 的变量是 A 和 B.C */
LIST = 'A B C'
DROP (LIST) F    /* DROP 的变量是 A、B、C 和 F,但是 LIST 变量的值不变 */
```

3.3.7 INTERPRET 关键字

INTERPRET 指令可以动态地翻译指令并执行，通过在关键字后面添加任意指令内容，包括 INTERPRET 本身，来指定要解析和执行的命令。如果用到 DO-END，IF-THEN-ELSE 等指令要注意成对匹配，同时，如果在非循环结构内，也不可以使用 ITERATE 和 LEAVE 指令。INTERPRET 指令常用在一些特殊情况下，如许多语句需要同时被翻译，或者有些语句要动态翻译等。

```
INTERPRET SAMPLE '= 4'
/* REXX 首先创建出字符串"SAMPLE = 4"，然后执行将 4 赋值给 SAMPLE 变量 */

SAMPLE = 'DO 3; SAY "HELLO WORLD!"; END'
INTERPRET SAMPLE
```

```
/* 输出三行"HELLO WORLD" */
```

通常对于语言程序的执行可以采用先编译成机器码再执行的方法，一次性完成全部代码的转换然后全部执行；还可以采用边翻译边执行的方法，每次翻译一条语句，执行后再翻译下一条语句。REXX 程序可以用任一方法执行，但是全部编译再执行的方法可以保留生成的可执行代码，避免每次运行时重复的翻译过程。

3.3.8 NOP 关键字

NOP 作为空语句，常用于 THEN 和 ELSE 子句，表示不执行操作。如果不用 NOP 关键字而是使用分号占位，则该子句将被忽略，直接跳到下一句执行，易产生语法错误，如下例。

```
SELECT
WHEN A = C THEN NOP
WHEN A > C THEN SAY 'A > C'
OTHERWISE SAY 'A < C'
END
/* 如果不使用 NOP,则将把第二个 WHEN 子句当做第一种情况的处理结果 */
```

3.3.9 NUMERIC 关键字

NUMERIC 关键字用于控制算术运算的方式，包括 NUMERIC DIGITS、NUMERIC FORM 和 NUMERIC FUZZ。

1. NUMERIC DIGITS

NUMERIC DIGITS，参数是数值或计算术值的表达式，用于控制计算的精度，即有效位的数量，默认是 9 位有效值，可以指定一个有效的正整数设置有效值，但要比当前状态下 NUMERIC FUZZ 的设置大。

对 DIGITS 的设定没有范围限制，只要在系统的存储空间内即可，但高精度的计算常常会消耗更多的时间，所以一般情况下建议使用系统默认精度。可以使用系统内置函数 DIGITS 来得到当前的 NUMERIC DIGITS 设置。

```
NUMERIC DIGITS 5   /* 设置当前的表示精度为 5 位 */
SAY 54321*54321    /* 显示输出为 2.9508E+9 */
```

2. NUMERIC FORM

NUMERIC FORM 参数为 SCIENTIFIC、ENGINEERING 或 VALUE（expression），expression 的计算结果是 SCIENTIFIC 或 ENGINEERING，用于控制计算结果的指数表示方法，可以是科学计数法（SCIENTIFIC），即在小数点之前只有一位非零的整数，也可以是工程计数法（ENGINEERING），即小数点左边的数字可以有 1～3 位有效数字，而 10 的乘方的指数必须是 3 的位数。默认情况下使用科学计数法表示，可以通过内置函数

FORM 来得到当前 NUMERIC FORM 的设置。

```
NUMERIC FORM SCIENTIFIC  /* 设置使用科学计数法 */
123.45 * 1e11 -> 1.2345E+13
```

3. NUMERIC FUZZ

NUMERIC FUZZ 用于在数值比较时，设置可忽略的位数，默认情况是 0，也就是每一位都参与比较，用户也可以自己设置一个非负整数，并且要比 NUMERIC DIGITS 的设置精度小。当进行数值比较时，会将数字按照（DIGITS-FUZZ）的精度来做减法，再与 0 比较。可以通过内置函数 FUZZ 来得到当前 NUMERIC FUZZ 的设置。

```
NUMERIC DIGITS 5
NUMERIC FUZZ 1          /* 设置比较精度为 4 位,与 NUMERIC DIGITS 4 作用相同*/
SAY 4.9999 = 5          /* 结果为 "1" */
```

3.3.10 OPTIONS 关键字

OPTIONS 关键字向编译器发出请求或设置参数，如设置编译选项或定义字符集等。通过提供一个表达式来设定，表达式会被逐字解析，如果可以识别就进行相应的设置，如果不能识别就转交给其他编译器。通常可以识别的参数如下（常作为字符串给出）。

（1）ETMODE：告诉编译器要对含有 DBCS 字符的语句进行有效性检查，即可以使用 DBCS 字符集，该参数需要在程序的最开始设置，并且后续的设置都将被忽略，否则会返回错误信息 IRX0033I。如果含有 OPTION 指令的表达式是一个外部函数的调用，并且调用的函数中含有 DBCS 编码的字符串，那么函数名称需要加引号，这样可以保证函数中的 DBSC 字符串被识别，而不是在调用前执行 OPTION 指令。通常情况下不建议用内部函数来设置 ETMODE，这样可能会导致在翻译程序中 DBCS 字符串时出现错误。

```
OPTIONS 'ETMODE'   /* 可在后面的代码中使用 DBCS 编码的字符 */
```

（2）NOETMODE：和 ETMODE 相反，不检查 DBCS 字符串的有效性，是 OPTIONS 的默认设置，如果不是程序的第一条指令将被忽略。

```
OPTIONS 'NOETMODE' /* 代码中的 DBCS 字符将无法识别 */
```

（3）EXMODE：告诉编译器当指令，运算或函数在处理含有 DBCS 数据的混合字符串时，用逻辑处理，不进行类型转换，保持 DBCS 数据的完整性。在函数或子例程中对该参数的设置会被保留到原程序中。

```
OPTIONS 'EXMODE'  /* 检查程序中混合字符串的有效性 */
'ab<cd'           /* 错误，没有结束符号*/
'ab<.a.b.c>       /* 检查正确 */
```

（4）NOEXMODE：设置在处理含有 DBCS 的混合字符串时按位处理，可能会破坏 DBCS 数据的结构，是默认设置。

```
OPTIONS 'NOEXMODE' /* 不检查程序中混合字符的有效性*/
```

3.3.11　SIGNAL 关键字

SIGNAL 关键字指令可用于控制对程序的跟踪调试或者改变程序的执行顺序。使用 ON 或者 OFF 来设置跟踪的状态，并通过参数指定需要跟踪的事件类型，具体的使用方法参见第 9 章。

通过 SIGNAL 指令来控制程序的执行顺序，方法是指定要跳转到的标签名称，可以是给定的字符串或者是表达式的计算结果（需用 VALUE），并且指定的标签名称需是大写，否则编译器无法找到对应的标签位置。当遇到 SIGNAL 指令时，停止当前命令的执行，包括 DO、IF、SELECT 和 INTERPRET 命令，跳转到程序中第一个符合条件的标签处，并且这种跳转是不可恢复的，对标签的搜索是从程序起始位置开始的，如果有名字相同的标签，则跳转到第一次出现的地方。

当程序找到了对应的标签位置，SIGNAL 指令所在的行数将被记录到 SIGNAL 变量中，以便在调试的时候找到发生跳转的位置。

在 SIGNAL 指令中，还可以使用 VALUE 关键字，用来在运行时决定程序跳转的位置，例如：

```
NAME = 'TENNIE'
CALL FUNC NAME, 7
⋮
FUNC: PROCEDURE
ARG LABEL .                    /* 通过传递参数来指定要跳转的位置 */
SIGNAL VALUE LABEL
⋮
TENNIE: SAY ARG(1) '!' ARG(2)   /* 显示 "TENNIE ! 7" */
```

3.3.12　UPPER 关键字

UPPER 关键字可以将变量的值转换成大写，一次可输入多个变量（简单变量或复合变量），按照输入顺序依次转换。同 TRANSLATE 内置函数可以实现同样的功能，但是 TRANSLATE 函数每次只能转换一个变量，UPPER 指令比较方便。

```
A = 'hello'; B = 'world'
UPPER A B
SAY A B     /* 显示"HELLO WORLD" */
```

如果在 UPPER 后使用了常量或者复合词干，将会报错；使用了未定义的变量不会产生错误或影响，但是如果设置了对 NOVALUE 的跟踪，就会引发异常。

3.4 REXX 命令

REXX 程序遇到一个表达式时，会首先按照指令来解析，如果无法执行，就认为这是一条命令，并将它发送给当前的宿主环境来执行。宿主环境执行了命令后，会把控制权交回 REXX 编译器，并设置变量 RC 来保存返回值，反应命令执行的情况。除此之外，还可能有一些附加的信息，如 ERROR 和 FAILURE 等。

ERROR 常常是可预期的，如定位字符串失败等，返回值为正。FAILURE 常常是不可恢复的，如命令无法执行或未找到相应命令等，返回值为负数。如果程序中开启了对 ERROR 和 FAILURE 的跟踪，或者 TRACE E 和 TRACE F 被设置，那么 ERROR 和 FAILURE 的发生将会影响到程序的正常执行顺序。

在 REXX 命令发送到对应的宿主环境之前会先进行解析，所以如果需要保留原始输入状态的部分要加引号。REXX 程序可以发出的命令主要有两种，TSO/E REXX 命令和其他宿主环境命令。本节主要介绍 TSO/E REXX 命令，关于宿主命令详见第 6 章。

3.4.1 TSO/E REXX 命令

在 TSO/E 环境下，提供了许多供 REXX 使用的系统命令，和传统的 TSO/E 命令（如 ALLOCATE、PRINTDS）有所不同，它只能用于 REXX 程序，在 CLIST 和 TSO/E READY 模式下不能使用，但有一些命令除外，如 EXECUTIL、HE、HI、HT、RT、TE 和 TS。

EXECUTIL 命令只能在 TSO/E 地址空间中运行，它可以在 CLIST 程序以及 TSO/E READY 和 ISPF 面板中使用；TS 和 TE 命令可以在任何地址空间中运行；HE、HI、HT、RT、TE 命令可以在 TSO/E 地址空间使用，用于对警示中断键（attention interrupt key）的回复。

REXX 命令可以提供的服务有：通过 I/O 控制对数据集的读写（EXECIO），操作数据栈（如 MAKEBUF、DROPBUF、QBUF、QELEM、NEWSTACK、DELSTACK、QSTACK），检查宿主命令环境（SUBCOM），改变程序执行（EXECUTIL）等。

3.4.2 TSO/E REXX 命令的执行

在执行 REXX 命令时，可以传递参数，如果参数是一个变量，则不能用引号，因为要先解析变量的值并替换原命令中的变量名称，然后传递到 TSO/E 环境执行，否则会直接将带引号的字符串不加修改地传出去，从而无法传递正确的数值，正确的用法如下：

```
NAME  = MYREXX.EXEC
"LISTDS" NAME "STATUS"
```

1. 交互模式

如果操作的 TSO/E 环境允许有用户提示，那么当输入一个 REXX 命令而未指定操作

内容或参数时，系统会提示用户输入，例如：

```
READY
LISTDS
ENTER DATA SET NAME -    /* 提示输入数据集的名称 */
```

用户可根据需要打开提示设置，显式地输入 PROMPT 参数，例如：

```
EXEC MYREXX.EXEC(TEST) EXEC PROMPT
```

或者通过 PROMPT 函数来指定，例如：

```
SAVE = PROMPT('ON')      /* SAVE 保存当前 PROMPT 的设置 */
```

还可以用同样的方法关闭 PROMPT。

```
X = PROMPT('OFF')
SAY PROMPT()             /* 读取当前的 PROMPT 设置 */
X = PROMPT(SAVE)         /* 用原来保存的 PROMPT 值来设置现在的 PROMPT */
```

同时，如果在数据栈中有元素存在，并且用户输入的命令需要参数，那么系统将用数据栈中的元素来完成交互，所以要谨慎使用。

2. 显式调用和隐式调用

除了通过外部例程的方法执行 REXX 程序外，还可以通过在一个 REXX 程序中使用 EXEC 命令来显式地调用其他的 REXX 程序，通过 RETURN 或 EXIT 命令返回到调用程序，返回值保存在特殊变量 RC 中。

通常，当所调用的 REXX 程序和主程序不在同一个分区数据集中或者主程序和被调用程序不在系统数据集 SYSEXEC 和 SYSPROC 中时，会采用显式调用。

```
"EXEC MYREXX.EXEC(TEST) '20 A'"
SAY 'THE RESULT IS' RC
```

也可以隐式地调用 REXX 程序，即不使用 EXEC 命令，而是直接输出要执行的程序的名字和可能用到的参数，用引号括起来，程序名前面可以用%标记，这样能够缩短搜索的时间，也可以不标记。但此时要求主程序和被调用程序都要在 SYSEXEC 或 SYSPROC 中。

```
"%TEST 20 A"
```

3.4.3　常用的 TSO/E REXX 命令

本节主要列举一些常用的 REXX 命令。REXX 命令是 TSO/E 环境的保留字，不能用来命名 REXX、CLIST 和其他可执行模块。

1. 数据栈相关命令

关于数据栈的操作命令在 TSO/E 和非 TSO/E 环境中都能使用。如果在非 TSO/E 环境中同时有多个环境运行，只有创建数据栈的环境可以操作相应的数据栈，其他环境则不能。

1）NEWSTACK 和 DELSTACK 命令

NEWSTACK 命令用于创建一个数据栈，并且后续的操作都将针对这个新建的数据栈而原来的数据栈会被隐藏或隔离出来，在新的数据栈被删除后，才能使用原来的数据栈和里面的数据。一次可以创建多个数据栈，但是只有最新创建的可以被使用。

DELSTACK 命令用来删除最新创建的数据栈和里面的全部元素，用 QUEUED 函数可以得到当前使用的数据栈中元素的数量。

```
"NEWSTACK"    /* 创建新数据栈 */
PUSH elem1
PUSH elem2
⋮
"DELSTACK"    /* 删除之前创建的数据栈 */
```

2）MAKEBUF 和 DROPBUF 命令

MAKEBUF 命令用于在数据栈上创建一个新的缓冲区。在数据栈创建时，拥有一个 buffer0 缓冲区，MAKEBUF 创建新的缓冲区并返回编号，保存在 RC 中，如 1、2 等。

DROPBUF 命令可以删除最新创建的缓冲区并清空数据，也可以通过指定缓冲区的编号来删除一个特定的缓冲区和在它之后创建的所有缓冲区。如果使用 DROPBUF 0 命令，数据栈上所有由 MAKEBUF 创建的缓冲区和数据都将被清空，但是 buffer0 会保留，不会被删除（内部数据会清空），因为它不是 MAKEBUF 命令创建的。该命令的执行结果会存放在 RC 中，0 表示删除成功，1 表示指定的缓冲区编号不是一个有效的数值，如 DROPBUF A 命令，2 表示指定的缓冲区编号不存在。

```
"MAKEBUF"     /* 创建缓冲区 */
SAY 'The number of buffers created is' RC  /* RC = 1 */
PUSH elem1
PUSH elem2
"MAKEBUF"     /* 创建新缓冲区 */
PUSH elem3
⋮
"DROPBUF"     /* 将第二次创建的缓冲区删除 */
⋮
```

3）QSTACK 命令

QSTACK 命令用于返回先用程序中存在的数据栈数量，包括 buffer0 在内。如果未使用 MAKESTACK 命令，RC 的值为 1。该命令可以被程序以及它所调用的函数和子例程调用。

```
"NEWSTACK"      /* 创建数据栈 stack 2 */
⋮
"NEWSTACK"      /* 创建数据栈 stack 3 */
⋮
"DELSTACK"      /* 删除数据栈 stack 3 */
"QSTACK"
SAY 'The number of data stacks is' RC     /* RC = 2 */
```

4）QBUF 命令

QBUF 命令用来计算新创建的数据栈上建立的缓冲区数量，返回值存放在 RC 中。如果未使用 MAKEBUF 命令创建缓冲区，QBUF 命令返回 0。

```
"MAKEBUF"     /* 创建缓冲区 */
⋮
"MAKEBUF"     /* 创建缓冲区*/
⋮
"DROPBUF"     /* 删除后创建的缓冲区 */
"QBUF"
SAY 'The number of buffers created is' RC /* RC = 1 */
```

5）QELEM 命令

QELEM 命令用于计算新创建的数据缓冲区中元素的数量，如果未使用 MAKEBUF 创建缓冲区，则 RC 的值为 0，与当前数据栈中元素的数量无关。

```
"MAKEBUF"
PUSH one
PUSH two
PUSH three
"QELEM"
SAY 'The number of elements in the buffer is' RC /* RC = 3 */
```

2. EXECIO 命令

EXECIO 命令用于处理数据集的输入输出。可以一次从数据集中读取信息到数据栈中进行顺序处理或者放到一组变量中进行随机访问等操作。同样，也可以将数据栈或者变量的信息写回数据集中。EXECIO 可以对数据集进行增、删、查、改等许多操作，但是不支持对有跨域记录、磁道溢出或者未定义记录格式的文件进行 I/O 操作。

可以执行 I/O 操作的数据集可以是顺序数据集或者是分区/扩展分区数据集的一个成员，在对数据集进行 I/O 操作之前，要先通过命令将数据集分配给一个文件，EXECIO 本身命令不具有分配功能。

对于未指定 LRECL,RECFM,BLKSIZE 的文件，EXECIO 会按照默认值进行处理。但是如果文件的分配信息已经设置，如 BLKSIZE 为 0，即使向文件写入数据（EXECIO DISKW），文件块的大小也不会被 EXECIO 重置。

当操作的数据集是一个被多用户共享的系统数据集时，要在使用 EXECIO 之前指定数据集的类型为 OLD，以便在 I/O 操作时可以独占使用数据集。在操作结束后，常常需

要手动关闭数据集，除非操作数据集的任务结束了，或者编译器停止工作。

在使用 EXECIO 时，后面的操作数（DISKW，STEM，FINIS，LIFO）建议用引号括起来，以防止操作和变量名混淆。例如，如果定义了一个名为 stem 的变量，并且使用了 EXECIO STEM 命令，如果未使用引号，那么 STEM 将会被识别为变量，并用其相应的值来替换。具体的 I/O 操作参见第 5 章。

```
"EXECIO * DISKR myindd 100 (FINIS"
/* 从第 100 行开始读取全部的数据到数据栈中 */
"EXECIO 0 DISKR myindd 100 (OPEN"
/* 打开数据集，从第 100 行开始，不读取数据到数据栈*/
```

3. EXECUTIL 命令

EXECUTIL 命令用来控制程序在 TSO/E 环境下的执行，可以用在 REXX 程序，CLIST 程序以及 READY 和 ISPF 面板中。同样也可以在一些使用 TSO 服务的高级语言中搭配使用 EXECUTIL 和 HI、HT、RT、TS、TE 命令，但是在 READY 和 ISPF 中，不能使用 HI、HT 和 RT，因为当前没有正在运行的 REXX 程序。

EXECUTIL 命令可以完成以下功能。

- 设置系统运行库（默认是 SYSEXEC）在程序载入后是否关闭。
- 控制 TRACE 指令的开启和关闭。
- 终止程序的翻译过程。
- 禁止和恢复程序对终端的输出。
- 修改程序包目录中的项目。
- 设置除了 SYSPROC 外是否还有搜索其他的运行库。

EXECUTIL 命令的使用注意以下事项。

- 所有的 EXECUTIL 操作是互斥的，也就是说，每次只能使用一个运算量（HI、HT、RT、TE、TS）。
- EXECUTIL 命令是基于编译过程的命令，即它的作用范围是当前执行该命令的环境，如当用户使用分屏，并且在其中一个 ISPF 面板上使用了 EXECUTIL TS 命令，只有这个面板上调用的程序会被跟踪，另一个 ISPF 面板上的程序则不会。

示例：假设用户刚编写了一个新的子例程 SUBRTN，想跟踪该子例程的执行过程，可以使用下面的代码在子例程开始处使用 EXECUTIL TS 打开跟踪，在子例程返回主调程序之后使用 EXECUTIL TE 结束跟踪。

```
/* REXX program */
MAINRTN:
⋮
CALL SUBRTN
"EXECUTIL TE"
⋮
EXIT
/* Subroutine follows */
```

```
SUBRTN:
"EXECUTIL TS"
⋮
RETURN
```

下面介绍 EXECUTIL 后面常用的操作数。

1）EXECDD（CLOSE）和 EXECDD（NOCLOSE）

EXECDD 用来设置在程序载入但是运行之前是否要关闭系统运行库，可以用 CLOSE 来关闭或者 NOCLOSE 来打开，命令的作用范围是整个程序的运行空间直到下一条 EXECDD 命令的修改。

EXECDD 作用的系统库是由模块名称表（module name table）中的 LOADDD 参数指定的，默认是 SYSEXEC。如果设定了 CLOSE，则在程序载入后立即关闭对应的库。如果 SYSEXEC 中的其他程序已经或正在执行，那么 SYSEXEC 库就会被打开，此时如果使用 EXECDD 命令可能产生其他结果，因为 SYSEXEC 文件需要在和打开它的程序相同的 MVS 任务级别中关闭，所以要在运行库程序之前设置 EXECDD。

2）SEARCHDD(YES/NO)

SEARCHDD 操作用来设置当隐式调用程序时是否要搜索系统库（默认是 SYSEXEC）。YES 表示先搜索 SYSEXEC，如果没有找到再搜索 SYSPROC；NO 表示只搜索 SYSPROC。可以利用 EXECUTIL SEARCHDD 命令来动态设定搜索的顺序。

```
EXECUTIL SEARCHDD(NO)    /* 只搜索 SYSPROC 库 */
```

3）RENAME

RENAME 操作用于改变函数包目录（function package directory）中的条目，目录用来记录该函数包中函数和子例程的信息，主要有函数名、入口地址、可执行模块的入口地址、函数代码的载入的位置，具体操作参数如下。

① NAME（function-name）替换目录中 func-name 的值，指定新的函数或子例程的名字。

② SYSNAME（sys-name）替换可执行模块中调用函数代码段的入口地址，如果未指定 SYSNAME，则不替换，原模块失效。

③ DD（sys-dd）替换存放程序代码的数据集，如果未设定 DD 参数，则 sys-dd 的值设为空。

```
EXECUTIL RENAME NAME(PARTIAL)   /* 使函数包中名为 PARTIAL 的函数条目失效 */
```

可以使用 EXECUTIL RENAME 命令通过替换目录表项来执行不同的代码，这样可以在不改变函数和子例程名字的情况下执行不同的操作。

4. 立即命令

立即命令（immediate command）通常用于在 TSO/E 环境下执行 REXX 程序，当输入中断键时进入中断模式（attention mode），之后系统会提示 IRX0920I 消息，此时可以输入立即命令来回应，主要有 HI（Halt Interpretation），HT（Halt Typing），HE（Halt

Execution），RT（Resume Typing），TS（Trace Start）和 TE（Trace End），其中 HE 不是 EXECUTIL 后的有效操作符。另外，TS 和 TE 命令可用于非 TSO/E 环境。

1）TS 和 TE

TS（Trace Start）启动 TRACE，TE（Trace End）结束 TRACE。如果在 READY 模式或 ISPF 面板中输入 EXECUTIL TS 命令，那么接下来输入的 REXX 程序，它们所调用的子程序以及目前可能正在执行的程序都将被跟踪。

```
READY
EXECUTIL TS
```

如果要终止跟踪程序，可以调用 EXECUTIL TE 命令，在 REXX 程序或 CLIST 程序中都可以使用，该命令可以终止对目前运行的所有 REXX 程序的跟踪。

2）HI

HI 命令用于停止对所有 REXX 和 CLIST 程序的翻译，包括调用该命令的程序本身。如果某个程序设置了对挂起（HALT）情况的跟踪，并且含有 EXECUTIL HI 命令，那么这个程序以及它所调用的所有子程序会停止翻译，但是同时运行的其他程序将不受影响。如果一个 REXX 程序进入了无限循环，可以通过按下中断键，通过系统提示，输入 HI 命令来停止当前指令的执行来终止循环。

3）HT 和 RT

HT 命令用于挂起终端的输出，如 SAY 指令的输出结果，但是 REXX 程序继续运行，错误信息还是会显示，只有常规的输出被挂起，直到所有程序运行结束或者调用 EXECUTIL RT 命令来恢复终端输出。RT（Resume Typing）命令可以恢复之前挂起的输出。

HT 和 RT 指令对 CLIST 程序和其他命令无效。可以在 REXX，CLIST 程序和程序中调用 TSO 服务来执行 HT 和 RT 命令，但是在 READY 和 ISPF 下无效，因为并没有正在运行的 REXX 程序。终端输出被挂起的程序包括执行 HT 命令本身和其他正在运行的 REXX 程序。

在执行下例程序的循环语句时，每次输出一条语句。如果按下中断键（PA1 键），并输入了 HT 命令，那么终端将停止输出，当循环结束时，通过使用 EXECUTIL RT 命令来终止挂起过程，此时，循环中未输出的那些语句将重新输出到终端。

```
DO i = 1 to 10000
⋮
SAY 'The outcome is'...
END
"EXECUTIL RT"
```

4）HE 命令

HE 命令用于终止 REXX 程序的执行，该命令只对在 TSO/E 环境下执行的程序有效。它常常用于终止那些用其他语言编写的外部函数或子例程。如果要终止 REXX 程序，建议用 HI 命令。

如果在嵌套的程序中使用 HE 命令，它在 TSO/E 和 ISPF 环境下的工作方式略有差别。如程序 A 调用程序 B，并且在程序 B 执行期间，用户使用了 HE 命令挂起当前程序，那么，如果程序 A 是在 ISPF 环境下被调用的，HE 命令会停止 A 和 B 两个程序的执行；如果程序 A 是在 TSO/E READY 环境下被调用的，HE 命令只会停止程序 B 的执行，而上一级程序 A 是否停止还取决于它调用程序 B 的方式：如果程序 A 是通过 EXEC 指令调用程序 B，那么 HE 命令不会停止程序 A 的执行，而是将 RC 赋值 12，并从 EXEC 语句后继续执行；如果程序 A 是通过其他方式调用的程序 B，如 CALL 指令或函数调用，那么 HE 命令停止程序 B 后，将控制权返回到程序 A，同时产生一个 SYNTAX 的特定事件，表明子例程或函数执行的失败情况。

如果使用 HE 命令停止一个外部的子例程、函数或宿主命令，那么这些外部的例程在停止后将不再有机会恢复控制权来完成函数使用资源等的释放工作，需要由恢复程序 ESTAE 等来执行函数等的清理工作。

例如，在 TSO/E 环境下执行 REXX 程序，该程序调用了一个外部函数，而外部函数进入了无限循环，此时可以按下中断键，并输入 HE 命令来终止外部函数的循环。

5. SUBCOM 命令

SUBCOM 命令通过设置 RC 的值来检测某个宿主环境是否可用。如果 RC 为 0，标识可用，为 1 表示不可用。在非 TSO/E 环境下，可用的宿主环境有 MVS（默认）、CPICOMM、LU62、LINK、ATTACH、LINKPGM、ATTACHPGM、LINKMVS 和 ATTCHMVS。在 ISPF 环境下，可用的宿主环境有 TSO（默认）、CONSOLE、ISPEXEC、ISREDIT、CPICOMM、LU62、MVS、LINK、ATTACH、LINKPGM、ATTACHPGM、LINKMVS、ATTCHMVS。

```
"SUBCOM ISPEXEC"
IF RC = 0 THEN
ADDRESS ISPEXEC
ELSE NOP
```

3.5　程序控制流

通常程序执行时，是按照指令的编写顺序执行的，也可以通过一些 REXX 指令来改变指令执行的顺序，如跳过指令、重复执行等。这些指令可分为条件控制指令、循环指令和中断指令。

3.5.1　条件控制语句

条件控制语句通过设定至少一个判断条件，通过它的真假值来选择相应的执行路线。条件指令分为两类：二选一和多选一。

（1）IF/THEN/ELSE 指令，用于两条支路的选择。

用法：

```
IF expression THEN instruction
ELSE instruction
```

如果将全部指令写在一行内，需要在 ELSE 之前加分号隔开，即 IF expression THEN instruction；ELSE instruction。如果在 THEN 或 ELSE 后没有执行内容时，最好用空子句 NOP 来填充；如果有多条需要执行的指令，可以用 DO-END 来包括全部执行集合，否则编译器将只执行 ELSE 后的第一条指令。

同时该指令可以嵌套，但要注意 IF-ELSE 与 DO-END 的配对使用，例如：

```
IF weather = fine  THEN
DO
SAY 'What a lovely day!'
IF tenniscourt = free  THEN
SAY 'Shall we play tennis?'
ELSE
NOP
END
ELSE
SAY 'Shall we take our raincoats?'
```

（2）SELECT/WHEN/OTHERWISE/END 指令，用于多路选择，如图 3-1 所示。

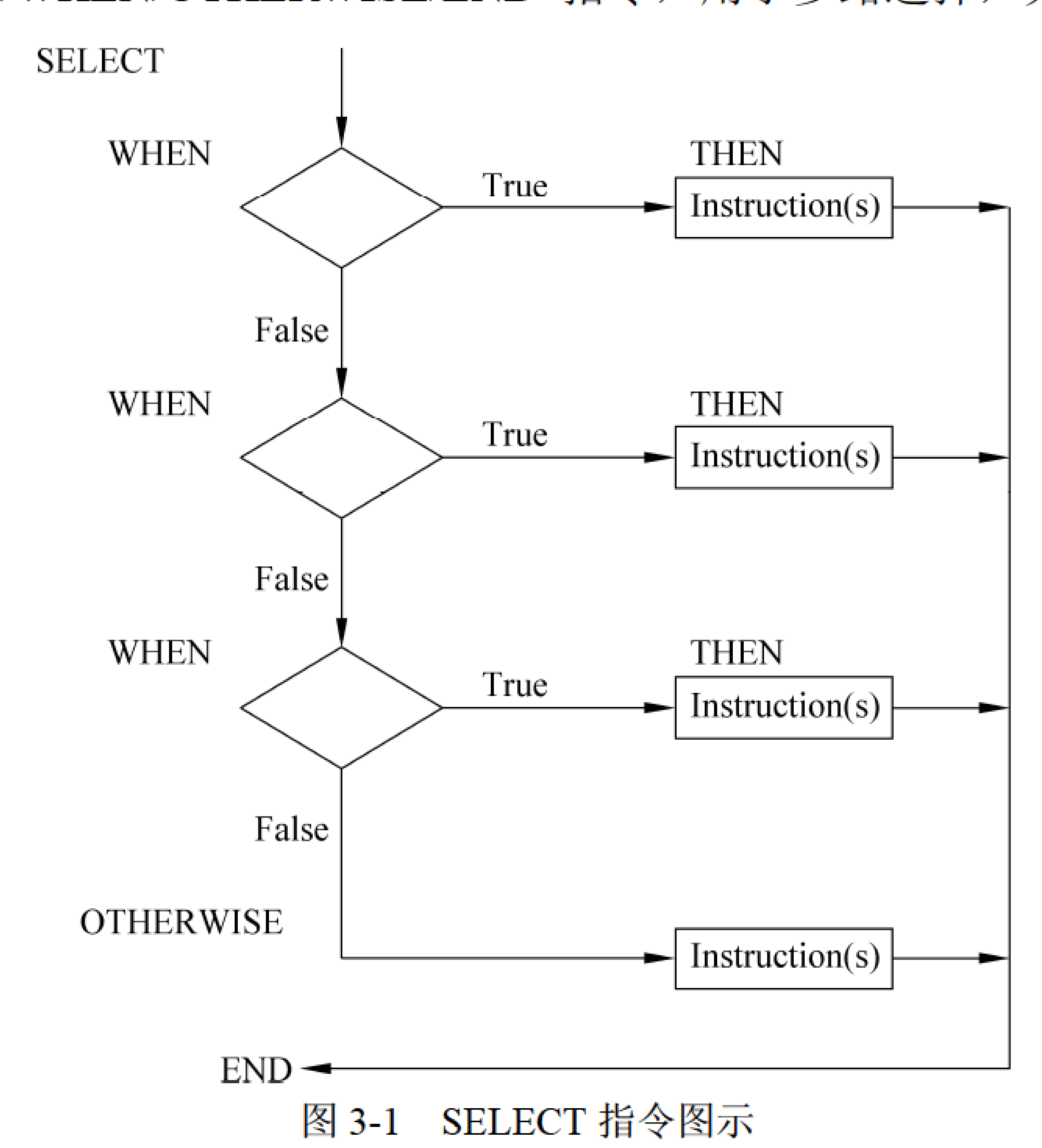


图 3-1　SELECT 指令图示

用法：

```
SELECT
WHEN  expression  THEN  instruction
```

```
WHEN  expression  THEN  instruction
:
OTHERWISE
instruction(s)
END
```

编译器通过 WHEN 后的条件来选择执行的指令，并且在找到一个条件为真的表达式后就跳出 SELECT 指令，忽略后续的所有内容，如果没有一个 WHEN 后的表达式为真，就执行 OTHERWISE 后的指令。同样，这里可以使用 DO-END 来包括许多条指令，用在 OTHERWISE 后的指令可省略 DO-END 结构。

```
SELECT
WHEN weather = fine  THEN  SAY ' What a lovely day!'
WHEN tenniscourt = free  THEN  SAY ' Shall we play tennis?'
OTHERWISE
SAY ' Shall we stay at home?'
END
```

3.5.2　循环控制语句

循环控制语句可重复执行一套指令，通过执行特定的次数或者条件判断来确定循环的终止。循环控制指令主要分为两类：重复循环和条件循环。所有循环语句都是在 DO-END 结构内，循环可以嵌套，注意 DO 和 END 的匹配即可。

用法：

```
Do repetitorClause
Instruction(s)
End
```

1. 重复循环语句

可以用数值常量或者变量来控制循环次数。当使用变量时，默认情况下每次变量的值增加 1，也可以通过 BY 来更改每次的增量值，通过 FOR 来控制最大循环次数。

```
Do i = 1 to 10 BY 2 FOR 2
SAY 'HELLO WORLD. '
END
/* 结果将输出 2 行 HELLO WORLD */
```

2. 条件循环语句

条件循环包括 DO WHILE 字句和 DO UNTIL，二者的区别是 DO WHILE 子句在第一次进入循环体前即进行循环条件检测，仅当条件表达式为真时继续循环；DO UNTIL 子句在执行后进行条件检测，仅当条件表达式为假时继续循环。因而后者至少执行一次。

```
QUANTITY = 20
DO NUMBER = 1 TO 10 WHILE QUANTITY < 50 /*(UNTIL QUANTITY > 50)*/
```

```
QUANTITY = QUANTITY + NUMBER
SAY 'QUANTITY = 'QUANTITY
END
/* 两条语句的输出结果相同 */
```

3. DO FOREVER 结构

DO FOREVER 循环可以实现无限循环，例如，要读一个文件中的数据，到文件结尾再停止，或者由用户通过特定的输入停止程序，这些情况下就比较适合无限循环，当满足结束条件时，再通过 EXIT 停止循环。可以用 DO-WHILE 来实现 DO-FOREVER 的功能。

4. LEAVE 和 ITERATE 命令

LEAVE 指令可以立刻终止循环，跳到 END 语句之后执行，ITERATE 指令终止当次的循环，回到 DO 语句，开始新一次的循环。例如：

```
DO i = 1 TO 5
   IF i = 3 THEN
        ITERATE
   ELSE
        SAY i
END
/* 输出为 1 2 4 5 */
DO outer = 1 TO 2
   DO inner = 1 TO 2
      IF inner > 1 THEN
         LEAVE inner /*跳出内循环*/
      ELSE
         SAY 'INNER'
   END
SAY 'OUTER'
END
/*输出为 INNER OUTER INNER OUTER */
```

上例中 LEAVE inner 中 inner 为循环变量，在循环嵌套中，REXX 允许用循环变量标示一个循环。上例中 LEAVE inner、ITERATE inner、LEAVE outer、ITERATE outer 都是合理的用法。另外一个示例如下。

```
Do outer = 1
   ......
   Do inner = 1
      ......
      If ... Then Iterate inner
      If ... Then Iterate outer
      If ... Then Leave inner
   End inner
   ......
End outer
```

5. 死循环处理

如果程序陷入了无限循环，可以通过中断键（attention interrupt key）停止循环，此时用户会得到一条 IRX0920I 的消息，回复 HI。如果循环未停止，可再次按中断键，回复 HE。

HI 不会停止由程序调用的非 REXX 的外部函数、子例程和宿主命令，直到程序返回 REXX 时才会响应 HI 请求。HE 可以停止以上这些情况下的无限循环，但是不会停止调用这些程序的 REXX 程序，在 ISPF 环境下运行时除外。同时 HE 不会改变 HI 请求的结果，也就是说，如果先输入了 HI 但未成功停止，然后又输入了 HE 来终止循环，此时返回调用的 REXX 程序内，之前 HI 的请求将会被跟踪到。

3.5.3 中断语句

中断语句可告知编译器退出整个程序或者程序的一部分，然后跳转到标识（label）处，或者执行子例程。中断循环的指令包括以下 3 类。

1. EXIT

EXIT 指令使程序无条件终止并且返回到程序调用处，如果该程序是由 ISPF 面板的 PROC 部分调用的，EXIT 指令将程序返回 ISPF 面板。与此同时，EXIT 会向调用者返回一个值。如果程序是另一个 REXX 程序的子例程，则返回值存储在特殊变量 RESULT 中；如果是一个函数，则返回值存储在函数调用的表达式中；其他情况下，返回值存储在特殊变量 RC 中。返回值可用于表达式或常量的计算。

2. CALL/RETURN

CALL 指令通过调用内部或者外部子例程来中断现有程序，其中，内部子例程是调用程序的一部分，通过标签标识调用位置（在 CALL 指令后面），而外部子例程则是另外一个 REXX 或其他系统支持的程序，通过程序名来标志。当调用程序结束后，通过 RETURN 指令返回到原调用处，继续执行原程序，如图 3-2 所示。

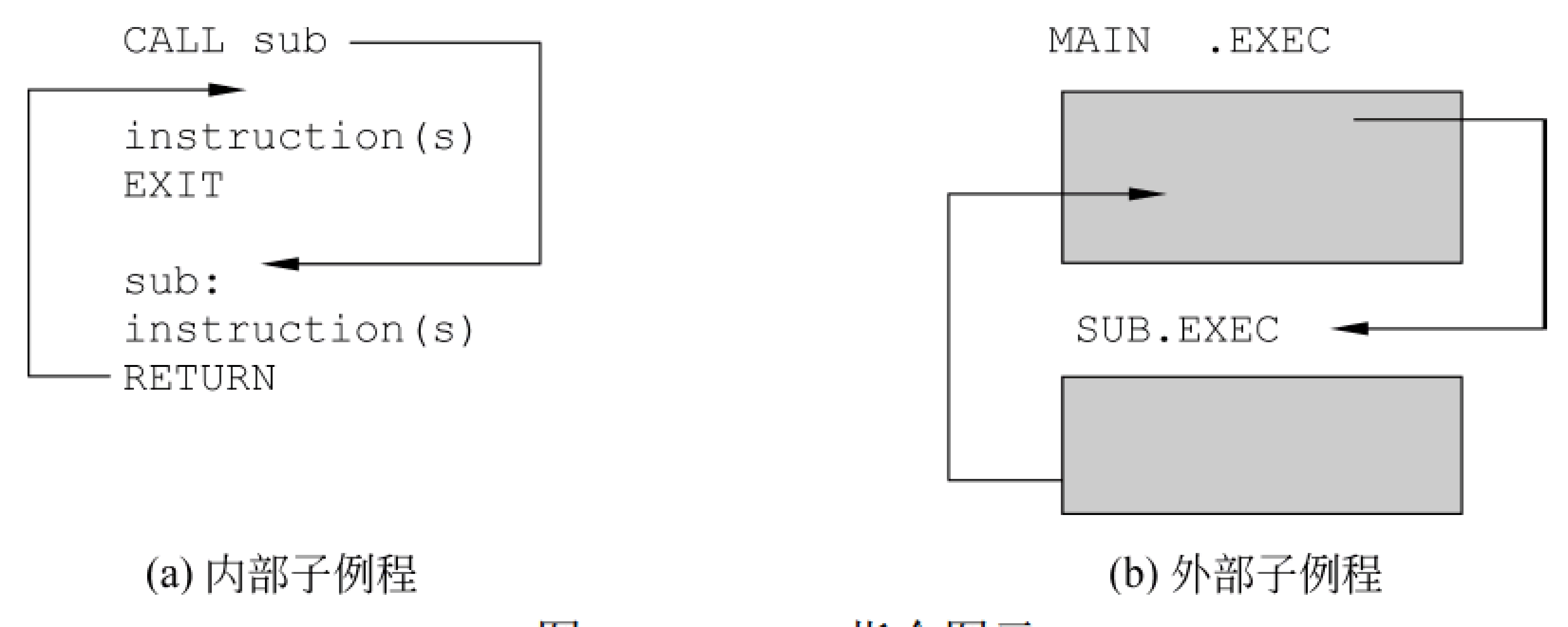


图 3-2 CALL 指令图示

3. SIGNAL

SIGNAL 的功能与 CALL 类似，可以跳转到目标标签处，并且标签的位置可以置于程序的任意位置，不必放在 SIGNAL 指令之后。与 CALL 指令不同的是，SIGNAL 指令不会返回到指令的调用处。即如果在循环内使用 SIGNAL 指令，则程序从 SIGNAL 后的

标签处继续执行，而原有的循环自动终止；如果在一个子例程内部使用 SIGNAL 指令，则该子例程不会再返回到其调用处。

SIGNAL 指令主要用于程序的测试或特殊情况的处理，而不是通常意义上的跳转，因为它无法返回，例如：

```
DO UNTIL EXPRESSION1
    INSTRUCTION
  IF EXPRESSION2 THEN
        SIGNAL JUMP
    ELSE
        INSTRUCTION(S)   /* 如跳转，这些指令将不会被执行 */

JUMP:
INSTRUCTION(S)
```

第 4 章

函数和子例程

函数和子例程是由一系列指令组成的，是可以用来接收、处理和返回数据的程序片段，包括内部例程和外部例程两种。内部例程是和调用程序在同一个源文件内，用标签标识函数体，并且只能被当前程序调用；外部例程是一个程序或可执行模块，可以被其他的程序调用，为了保证可以找到外部例程的位置，调用程序和被调用的例程都要存放在系统库 SYSEXEC 和 SYSPROC 中，或者放在同一个分区数据集的两个成员中。函数和子例程的指令内容相近，但是在调用方式和返回值方面不尽相同，所以要根据不同的需要来选择调用函数还是子例程。

4.1 函数的编写和调用

简单地说，函数是能够接受数据、处理数据并返回数值以完成某些特定功能的指令序列和代码片段，在 REXX 中，函数主要包括了以下几种。

（1）内置函数（built-in function）：即内建于 REXX 语言内部的功能函数，REXX 语言强大的原因之一就是功能丰富的内置函数能极大地方便用户编程。

（2）用户函数（user-written function）：即用户自己编写的功能函数，用户函数可以在当前 REXX 程序（exec）内部以特定标签标识，称之为内部函数；亦可以位于主调程序的外部，或者本身就是自包含程序，都称之为外部函数。内部函数通常指令数目少，参数传递快，调用时搜索函数体的速度快；外部函数通常功能较复杂，并且可以被许多程序多次调用。

（3）函数包（Function package）：由产品或者某些用户编写的特定功能函数或者子例程以打包的形式提供发布。例如，TSO/E 外部函数就是通过系统函数功能包提供。

函数的调用形式如下，注意在函数名和左括号中不能插入空格。

```
ReturnValue=functionName( [ Expression[,Expression [, ...]] )
```

函数的参数列表最多可包含 20 个参数表达式，以逗号分隔，参数表达式可以是空格、常量、符号名称、字符串等，因此 function( )、function(66)、function(X)、function(‘literal string’)、function(function(X))等都是合法的调用形式。

所有函数都要有返回值，返回值替换函数的位置，所以函数调用常用于表达式中，

不会单独出现在一条指令中。

4.2 子例程的编写和调用

子例程和函数类似，都是由能够接受数据、处理数据并返回值的完成某些特定功能的指令序列和代码片段组成。子例程在调用结束时，通过 RETURN 语句返回。和函数不同的是，子例程对于返回值的要求不是必需的。

子例程同样分为内部子例程和外部子例程两种，内部子例程指的是位于 REXX 程序（exec）内部标签标识，仅在本程序中使用的例程；外部子例程指的是存放于分区数据集中成员中的程序片段或者 REXX 程序，可被其他多个 REXX 程序调用。子例程的调用方式如下。

```
CALL SubroutineName [Expression [, Expression [,...etc ] ]
```

在确定选择子例程或者函数实现相应功能时，主要考虑两点：一是返回值是否必需。如果不是，则可使用子例程实现。二是返回值是否作为表达式的一部分出现在指令中。如果是，则应考虑使用函数。事实上，很多子例程和函数的功能实现代码几乎相同，既可以被用作子例程，亦可被用作是函数调用，下例中的内置函数SUBSTR就是如此。

```
/* 内置函数 SUBSTR 的函数调用方式 */
x = SUBSTR('verylongword',1,8)      /* x 被赋值'verylong' */

/* 内置函数 SUBSTR 的子例程调用方式 */
CALL SUBSTR 'verylongword', 1, 8
x = RESULT                          /* x 被赋值'verylong' */

/* 以上两种方式均可实现对 SUBSTR 的调用 */
```

关于子例程的参数传递和返回值同前述的函数大致相同，读者可参考前文所述。下例给出一个使用了两种实现方式的实例。

```
/****************REXX*********************/
  CALL subroutine 'ab', 'cd', 'ef'
  SAY  result                      /* 显示结果为 abcdef */
  SAY  subfunc('ab', 'cd', 'ef')   /* 显示结果为 efcdab */
  RETURN 0

/************subroutin 子例程的实现*************/
  subroutine: PROCEDURE
  PARSE ARG string1, string2, string3
  RETURN string1 || string2 || string3

/************subFunc() 子函数的实现***************/
```

```
subfunc: PROCEDURE
PARSE ARG str1, str2, str3
RETURN str3||str2||str1
```

4.3 搜 索 顺 序

对函数和子例程的搜索顺序是：首先在程序内部搜索是否是内部函数或例程，其次查找是否是内置函数，最后是外部函数或例程，如图 4-1 所示。

- 内部例程：调用内部例程的时候，函数名称不能加引号。
- 内置函数：内置函数名称是大写的，所以在使用字符串表示时要注意大写。
- 外部函数和子例程：需要放正 SYSEXEC 和 SYSPROC 中，或和主程序放在一个分区数据集中。

```
CALL DATE
⋮
DATE: PROCEDURE
ARG IN
IF IN = '' THEN IN = 'STANDARD'
RETURN 'DATE'(IN)
```

上例中，调用函数 DATE 时，会先搜索程序内部，发现了 DATE 函数的定义后，会调用该函数，此时，系统自带的内置函数 DATE 将不会被调用。如果未在程序中定义 DATE 函数，那么将调用内置函数 DATE。

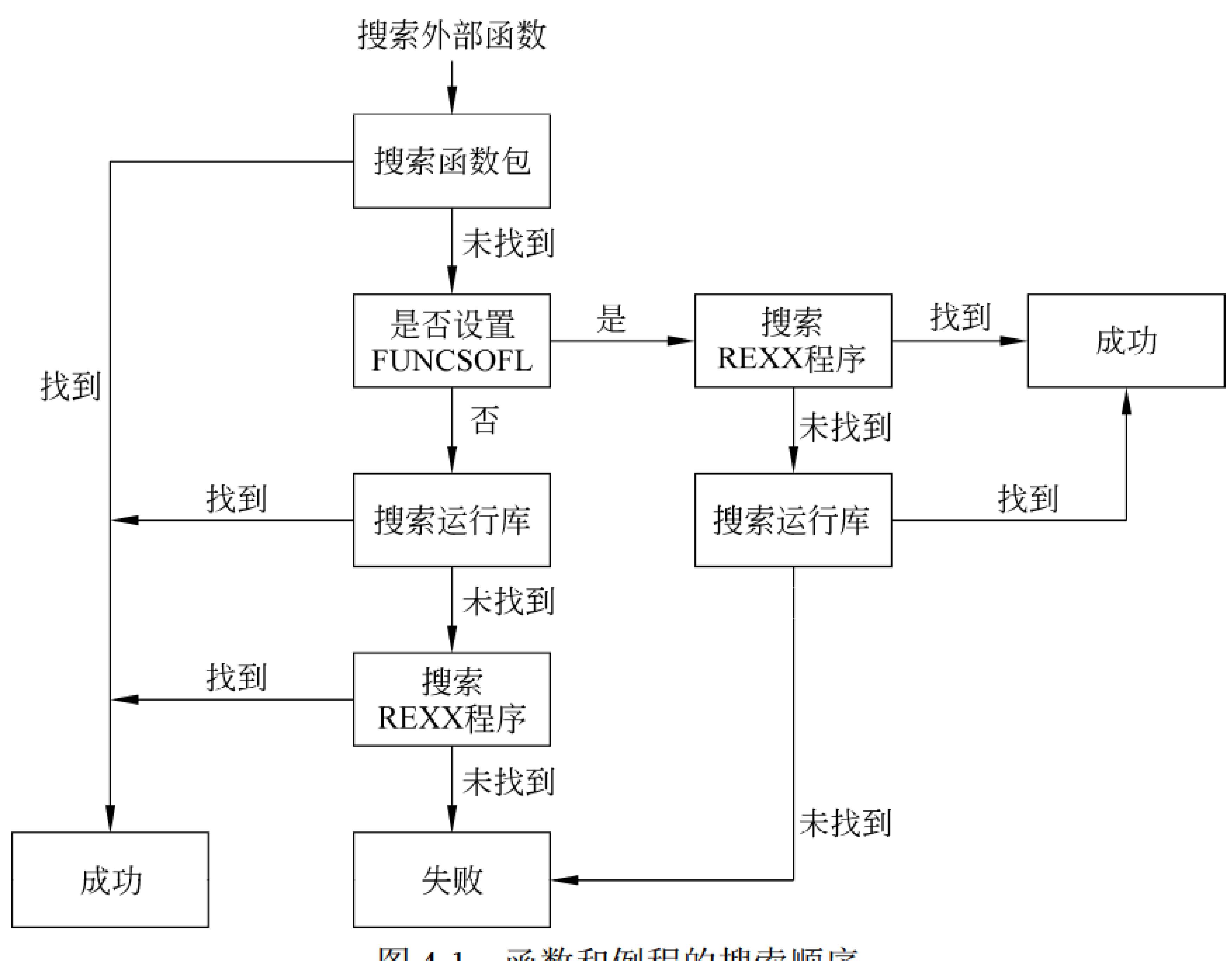


图 4-1　函数和例程的搜索顺序

对于外部函数和例程的调用，采用的顺序是，首先搜索函数包——用户自定义函数包、本地函数包和系统函数包。如果未找到函数，则系统会检查函数搜索标记 FUNCSOFL，判断是否要在搜索 REXX 程序前先搜索运行库。如果该标记设置为 OFF，则先检查运行库中是否有对应函数，没有再检查 REXX 程序。如果标识为 ON，则先检查 REXX 程序再检查运行库。默认情况下，FUNCSOFL 为 OFF。可以使用 TSO/E 环境下的 EXECUTIL RENAME 命令来改变函数包中的函数条目。

当搜索运行库时，顺序是：工作包区、ISPLLIB、任务库和前面的全部任务库、作业步库、链接包区（LPA）和链接库。

当搜索 REXX 程序时，顺序如下。

（1）含有调用函数的数据集。

（2）ALTLIB 命令设置的程序库。

（3）检查 NOLOADDD 的设置。如果 NOLOADDD 设置为 OFF，就搜索 SYSEXEC 中的数据集，（SYSEXEC 是系统默认的存储 REXX 程序的库），如果未找到函数，就搜索 SYSPROC 库，如果还没找到就停止搜索；如果设置为 ON，就直接搜索 SYSPROC，未找到就停止搜索。根据 FUNCSOFL 的设置，运行库可能未被搜索。

搜索 REXX 程序框图如图 4-2 所示。

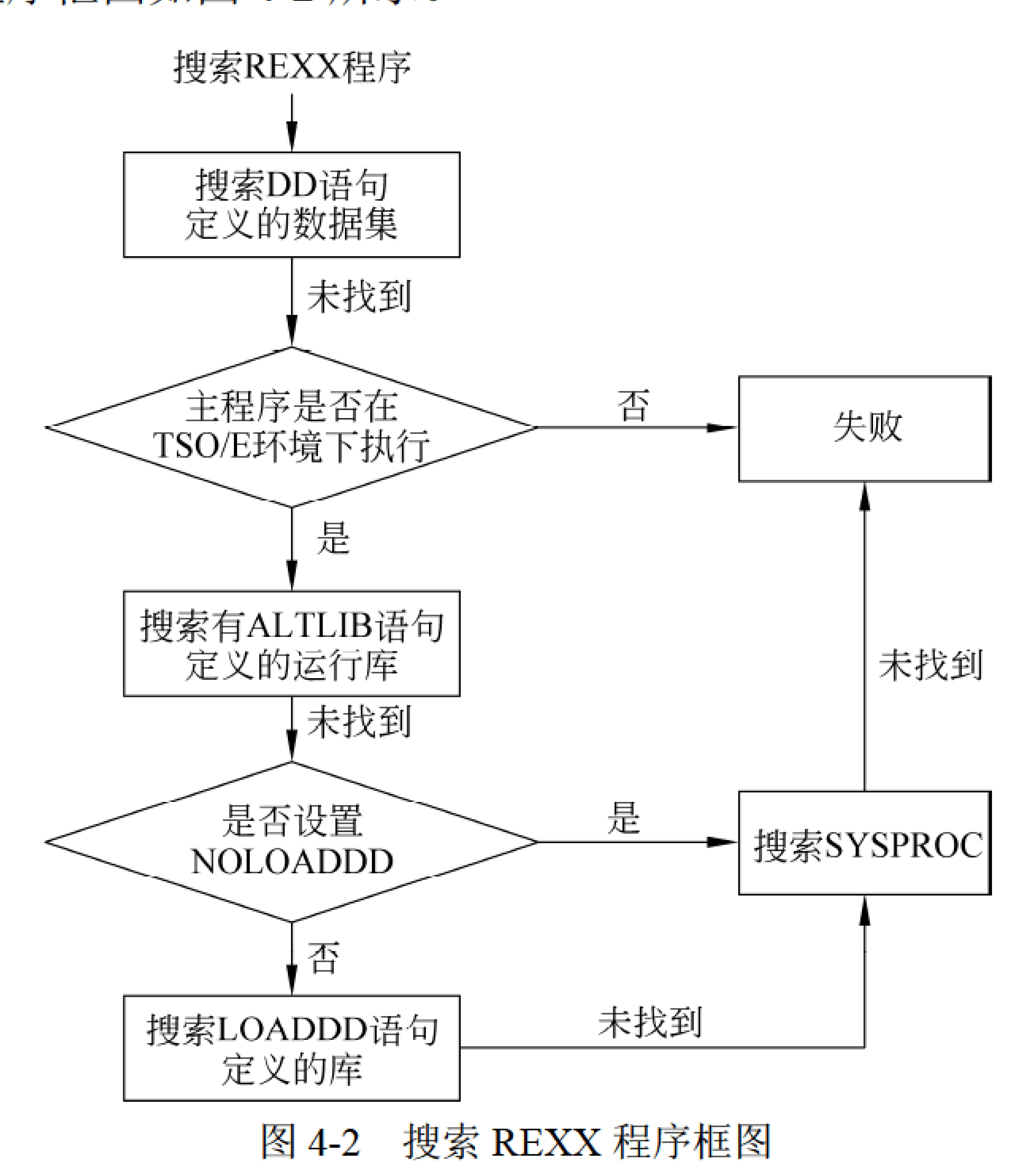


图 4-2 搜索 REXX 程序框图

4.4 参 数 传 递

REXX 主程序和内部函数、子例程之间可以共享变量，即通过使用相同的变量来传递参数值，并且不需要使用引号。但是对于外部的例程来说，就不能直接使用相同名称

的变量，需要通过调用时参数或数据栈等来传递数值。通常传递信息的方法有两种，变量共享和传递参数，下面分别简要介绍一下。

1. 变量共享

当主程序和被调用程序在同一个 REXX 程序中时，无论是在主程序部分还是子例程和函数中，只要是对这个变量的操作都是有效的，变量的结果会体现最后更新的操作。可以直接在被调用程序中使用该变量，也可以将子程序中的变量返回到主程序中，达到共享的目的，例如：

```
/************************REXX***********************/
firstName = "Amily"
lastName = "Wandor"
CALL subroutine
SAY name                    /*显示为"Amily Wandor"*/
EXIT

subroutine:
  name = firstName + " " + lastName
RETURN
/* 子例程直接使用 firstName 和 lastName 变量,并通过 name 返回结果 */
```

使用公共变量可能会产生各种问题，公共变量对所有子例程的暴露使程序逻辑变得混乱并难以维护。在 REXX 中，可以使用 PROCEDURE 命令解决：在子标签后添加 PROCEDURE 关键字，则规定所有子例程中出现的变量使用范围保护为子例程内部有效。当执行 RETURN 指令返回时，所有子例程中的参数值被 DROP 掉并恢复到调用之前，这样，即使子例程中使用了主调程序相同名的变量，也不会对主调程序产生影响，例如：

```
/*************************REXX***********************/
number1=10
CALL subroutine
SAY number1 number2     /* 显示结果为  10  NUMBER2  */
EXIT

subroutine: PROCEDURE   /* 子程序的变量被屏蔽,不再被共享 */
number1=7
number2=5
RETURN
```

有时，需要暴露子程序的某些变量，可以使用 PROCEDURE 的 EXPOSE 选项将某些变量暴露给主调程序。这样在子程序中对变量的操作会反映到主程序中，并且在 RETURN 后不会被复写。

如果不加括号，变量名可以用空格和逗号分开，例如：

```
/* REXX 主程序部分 */
J=1; Z.1='A'
```

```
CALL TOFT
SAY J K M                    /* 显示 "1 7 M" */
EXIT

/*REXX 子例程部分 */
TOFT: PROCEDURE EXPOSE J K Z.J
SAY J K Z.J                  /* 显示 "1 K A" */
K=7; M=3                     /* 注意: M 未被共享 */
RETURN
```

如果用括号将一个变量名括起来，则该变量名通常表示一个变量序列，该变量和它所包括的变量都会被共享，例如：

```
/* REXX 主程序部分*/
J=1; K=6; M=9
A ='J K M'                   /* 定义变量序列 */
CALL TEST EXIT

 /* REXX 子例程部分*/
TEST: PROCEDURE EXPOSE (A)  /* 暴露变量 A 和 J, K, M */
SAY A J K M                  /* 显示 "J K M 1 6 9" */
RETURN
```

2. 传递参数

另一种传递信息的方法是通过在调用函数和子例程时使用参数来传递数值，每次最多可传递 20 个参数，用逗号隔开。

```
CALL SUBROUTINE_NAME ARGUMENT1, ARGUMENT2, ARGUMENT3,...
FUNCTION(ARGUMENT1, ARGUMENT2, ARGUMENT3,...)
```

被调用的函数和子例程可以使用 ARG 指令接受参数，同样用逗号分隔，例如：

```
ARG arg1, arg2, arg3,...
```

ARG 和 CALL 的参数是通过位置对应，而不是名称，传过来的第一个数值给第一个参数，第二个数值给第二个参数，依此类推。

```
/****************************REXX************************/
PARSE ARG LONG WIDE
CALL PERIMETER LONG, WIDE
SAY 'THE PERIMETER IS' RESULT 'INCHES.'  /* RESULT = PERIM */
EXIT

PERIMETER:
ARG LENGTH, WIDTH            /* LENGTH = LONG, WIDTH = WIDE */
PERIM = 2*LENGTH + 2*WIDTH
RETURN PERIM
```

另一种接受参数的方式是使用内置的 ARG 函数，该内置函数返回了特定位置的参数值。如上例可写成：

```
LENGTH = ARG(1)
WIDTH = ARG(2)
```

子例程通过 RETURN 指令返回到主调程序，并且只能返回一个表达式。RETURN 指令中的表达式可以是数字（RETURN 8）、变量名（RETURN arg1 arg2）或者字符串（RETURN 'literal string'），甚至是运算表达式（RETURN num*12），返回表达式计算后的结果值放入到 REXX 的特殊变量 RESULT 之中，主调程序通过该变量即可访问返回值。

4.5　内 置 函 数

REXX 提供了功能丰富的内置函数，给用户编程带来了极大方便。此外，TSO/E 还提供了 6 个内置函数：EXTERNALS、FIND、INDEX、JUSTIFY、LINESIZE 和 USERID。

内置函数的使用示例如下。

```
/**************************REXX*************************/
SAY 'PLEASE ENTER A MEMBER NAME'
PULL NAME
IF LENGTH(NAME)>8 THEN       /* 如果名称字长大于 8 */
DO
   NAME = SUBSTR(NAME,1,8)  /* 截取名称头 8 个字符 */
   SAY 'THE MEMBER NAME YOU ENTERED WAS TOO LONG.'
   SAY NAME 'WILL BE USED.'
END
ELSE NOP
/* LENGTH()和 SUBSTR(),分别用于返回字符串长度以及截取子字符串 */
```

在使用内置函数的时候，要注意以下几点。

- 函数即使在没有参数的情况下，名字后面也要加括号，并且中间不能有空格。
- 内置函数不受 NUMERIC 命令影响，默认情况使用 NUMERIC DIGITS 9 和 NUMERIC FUZZ 0。
- 如果参数设置了长度或者选取了字符串中的起始位置，必须是一个有效的正整数。
- 如果最后一个参数是可选的，那么可以用逗号占位表示这个参数不设置或者取默认值，如 DATATYPE(1,)，和 DATATYPE(1)效果相同。
- 如果设置了字符串占位符，只能是长度为 1 的字符，参见格式函数 LEFT 的使用。
- 如果函数调用时可以使用字符串的首字母来代替特定字符串的内容，则该字母的大小写是任意的。

以下几节简要介绍各类函数的功能，详细用法和示例请参考 IBM 白皮书 *z/OS TSO/E REXX Reference*。

4.5.1 算术函数

算术函数主要是执行对数值的运算，遵循 NUMERIC 命令的设置，注意可选参数和默认值的使用，见表 4-1。

表 4-1 算术函数汇总

函　数	功能描述	示　例
ABS	求绝对值	ABS('-12.3') -> 12.3
DIGITS	返回 NUMERIC DIGITS 的设置	DIGITS() -> 8
FORM	返回 NUMERIC FORM 的设置	FORM() -> 'SCIENTIFIC'
FUZZ	返回 NUMERIC FUZZ 的设置	FUZZ() -> 0
MAX	求最大值，最多可指定 20 个参数	MAX(12,6,7,9) -> 12
MIN	求最小值，最多可指定 20 个参数	MIN(12,6,7,9) -> 6
RANDOM	产生随机数，可指定范围和种子	RANDOM(,,1983) -> 123
SIGN	返回数值符号（-1,0,1）	SIGN(' -0.307') -> -1
TRUNC	返回一定精度的实数值	TRUNC(127.09782,3)->127.097

上述函数中，DIGITS、FORM、FUZZ 函数的功能和对应关键字指令的功能相同，都是用来处理数据的表示方法和比较精度。

RANDOM 函数主要用于由系统产生一个随机数，但是这个随机数不是真正意义上的随机，而是通过用户指定一个在 0～999999999 范围的正整数作为随机种子，由系统通过数值计算产生随机数。相同的种子会得到相同的随机数。

```
RANDOM(MIN,MAX,SEED)
```

用户还可以指定随机数的范围，通过 MIN 和 MAX 来设定。若未指定 SEED 值，则系统会动态产生一个值作为种子，通常每次产生的动态值是不相同的。

4.5.2 比较函数

比较函数主要是比较字符串和数值是否相同或判断其类型，注意变量名要用引号，防止在执行函数前被相应变量值替换，见表 4-2。

表 4-2 比较函数汇总

函　数	功能描述	示　例
COMPARE	比较两个字符串，相同返回 0，不同返回第一个不同字母的位置，可使用占位符填补较短的字符串	COMPARE('abc','ak') -> 2 COMPARE('ab ','ab',' ') -> 0 COMPARE('ab ','ab','x') -> 3
DATATYPE	返回字符串的类型（NUM 或 CHAR），或判断是否是特定的类型	DATATYPE('12') -> 'NUM' DATATYPE('12.3','N') -> 1
SYMBOL	判断符号类型，BAD 表示无效，VAR 表示变量，LIT 表示常量或未初始化	SYMBOL('J') -> 'VAR' SYMBOL(2) -> 'LIT'

COMPARE 函数用于字符串的比较，具体用法如下。

```
COMPARE(STRING1,STRING2,PAD)
```

字符串需要使用引号，第三个参数表示占位符。占位符是指，对于两个长度不同的字符串，可以用占位符来填充长度较短的字符串，使之与长度较长的字符串长度一致，然后再进行比较。占位符从字符串的右侧开始填充。如表格中 COMPARE 函数的第二个例子，用空格来填充第二个字符串，这样两个字符串就相同了。

用 DATATYPE 判断字符串类型时，有以下有效类型，判断时可只用代表字母（单词中的大写字母）表示即可。

- Alphanumeric：只含有 a～z、A～Z 和 0～9
- Binary：只含有 0 或 1
- C：SBCS 和 DBCS 的混合字符串
- Dbcs：纯 DBCS 字符串，首尾是 SO 和 SI
- Lowercase：只含有小写字母 a～z
- Mixed case：含有字符 a～z 和 A～Z 的混合
- Number：是有效的 REXX 数值
- Symbol：只含有有效的 REXX 字符，大小写形式不限
- Uppercase：只含有大写字母 A～Z
- Whole number：在当前 NUMERIC DIGITS 设置下的有效的 REXX 整数
- heXadecimal：十六进制数，即只含有 a～f、A～F、0～9 和空格，或是空字符串

4.5.3　转换函数

转换函数用来进行数据类型转换，包括字符、二进制、十进制和十六进制，精度在 NUMERICDIGITS 设置范围内，见表 4-3。

表 4-3　转换函数汇总

函　数	功能描述	示　　例
B2X	将二进制转换成十六进制，可含空格或空串	B2X('1 1111 0000') -> '1F0'
C2D	将字符转换成十进制，如果指定显示精度，则按有符号数字处理	C2D('FF81'X) -> 65409 C2D('FF81'X,2) -> -127
C2X	将字符转换成十六进制	C2X('72s') -> 'F7F2A2'
D2C	将十进制转换成字符，可指定长度	D2C(129,2) -> ' a'
D2X	将十进制转换成十六进制，可指定显示长度	D2X(129,4) -> '0081'
X2B	将十六进制转换成二进制	X2B('C3') -> '11000011'
X2C	将十六进制转换成字符	X2C('F7F2 A2') -> '72s'
X2D	将十六进制转换成十进制	X2D('c6 f0'X) -> 240

4.5.4 格式函数

格式函数根据设置字符串中字符和空格的格式，其中 JUSTIFY 函数是 TSO 提供的，见表 4-4。

表 4-4 格式函数汇总

函　数	功能描述	示　　例
CENTER/ CENTRE	返回指定长度的字符串，居中显示，可使用填充符，若长度缩短，则从两侧截取	CENTER(abc,8,'-')-> '--ABC---' CENTRE('The blue sky',7) -> 'e blue '
COPIES	将字符串连接成重复指定次数的新串，次数为正整数或 0	COPIES('abc',3)-> 'abcabcabc' COPIES('abc',0)-> ''
FORMAT	格式化数字显示	FORMAT('3',4) -> ' 3'
JUSTIFY	将字符串中的单词用填充符均匀隔开，如果指定长度过短，就从右侧开始截短，并且去掉前导和后续的空格	JUSTIFY('The blue sky',14) -> 'The blue sky' JUSTIFY('The blue sky',9,'+') -> 'The++blue'
LEFT	返回指定长度的最左侧的子串	LEFT('abc d',8,'.')->'abc d...'
RIGHT	返回指定长度的最右侧的子串	RIGHT('abc def',5) -> 'c def '
SPACE	用指定数目的分隔符分隔单词，并去掉首尾的空格	SPACE(' a bc def ',2,'+') -> 'a++bc++def '

关于 FORMAT 的使用，具体介绍如下。

```
FORMAT(number,before,after,expp,expt)
```

FORMAT 可以对输入的数字进行一定的格式化设置，具体参数有 before、after、expp 和 expt。在这 4 个参数中，可以分别填入相应的非负整数，或省略取默认值等。

其中，before 和 after 分别用来设置整数和小数部分的位数，如果省略了这两个参数，则使用原数值表示。如果 before 的值没有原数字中的整数部分位数多，则报错；如果比实际的位数多，则用空格来填补高位。如果 after 的数值和实际小数部分的位数不相同，则或者舍入，或者用 0 来填充低位，设置 after 为 0 则会去掉小数部分。

```
FORMAT('1.73',4,0) -> ' 2'
FORMAT('1.73',4,3) -> ' 1.730'
FORMAT('-.76',4,1) -> ' -0.8'
```

expp 和 expt 参数用来设置数值的指数部分。expp 设置指数部分的位数，如果设置为 0，则不使用指数表示，而是简单的用小数点后的数字填充；如果 expp 数值不够大，则报错；如果 expp 不是 0 而实际的数值中指数部分是 0，则在结果中，会有（expp+2）个空格放在指数位置。expt 指定什么时候使用指数表示，规则是在整数或小数部分的位数分别超过了 expt 的一倍和两倍，那么就使用指数表示。如果 expt 为 0，则在有小数部分的时候就使用指数表示；如果 expp 为 0，则 expt 自动设置为 0。

```
FORMAT('12345.73',,,2,2) -> '1.234573E+04'
```

```
FORMAT('12345.73',,3,,0) -> '1.235E+4'
FORMAT('12345.73',,,3,6) -> '12345.73'
FORMAT('1234567e5',,3,0) -> '123456700000.000'
```

4.5.5　字符串操作函数

字符串操作函数用来设置字符串中的字符和空格的显示方式，其中 FIND 和 INDEX 是 TSO 提供的函数，见表 4-5。

表 4-5　字符串操作函数汇总

函　　数	功能描述	示　　例
DELWORD	从指定位置开始删除一定数目的用空格分开的单词，返回剩余字符串	DELWORD('Now is the time' , 2,2) -> 'Now time'
FIND	返回指定子串的首个单词在原字符串中的起始位置	FIND('now is the time','is the time') -> 2
INDEX	在第一个字符串中找第二个字符串的起始位置，可从指定位置开始查找	INDEX('abcabc','bc',3)-> 5 INDEX('abcabc','bc',6)-> 0
INSERT	将第一个字符串插入到第二个字符串中，可指定插入位置，插入的字符数量和填充字符，默认从开头插入，用空格填充	INSERT(' ','abcdef ',3) -> 'abc def ' INSERT('23','abc',5,6,'+') -> 'abc++23++++'
LASTPOS	返回指定子串在原字符串中最后一次出现的位置	LASTPOS(' ','abc def ghi') -> 8
LENGTH	返回字符串的长度	LENGTH('abcdefgh') -> 8
OVERLAY	用指定字符串覆盖原字符串中特定位置的字符	OVERLAY(' ','abcdef ',3) -> 'ab def '
POS	返回子串在原字符串中的位置	POS('day','Saturday') -> 6
REVERSE	将字符串首尾翻转	REVERSE('ABc.') -> '.cBA'
STRIP	除去首或尾的空格或指定字符，B 代表首和尾，L 代表首，T 代表尾	STRIP(' ab c ','L') -> 'ab c ' STRIP('12.70',,0) -> '12.7' STRIP('12.70',,1) -> ' 2.70'
SUBSTR	返回从指定位置开始的指定长度的子串，可指定填充符	SUBSTR('abc',2,6,'.') -> 'bc....'
SUBWORD	返回从指定位置的开始的指定长度的子串，以单词为单位	SUBWORD('Now is the time', 2,2) -> 'is the'
TRANSLATE	将一种字符翻译成对应规则的字符，默认是大写转换	TRANSLATE('abcdef ','12','abcd','.') -> '12..ef '
VERIFY	判断字符串是否只包含指定的字符，是返回 0，或返回不符合的字符位置	VERIFY('123','1234567890') -> 0 VERIFY('1Z3','1234567890')-> 2
WORD	返回字符串中指定位置的单词	WORD('I am a boy',3)-> 'a'
WORDINDEX	返回第 *n* 个单词的首字母的位置	WORD('I am a boy',3)-> '6'
WORDLENGTH	返回第 *n* 个单词的长度	WORD('I am a boy',3)-> '1'

续表

函　　数	功能描述	示　　例
WORDPOS	返回指定单词是原字符串中的第几个单词，可指定起始搜索位置	WORDPOS('be','To be or not to be',3) -> 6
WORDS	返回字符串中用空格分隔的单词数	WORDS(' I am a boy')-> 4

TRANSLATE 函数是用于将原字符串中的部分字符替换成新的字符，通过用户定义的转换规则来完成字符的变换，使用规则如下。

```
TRANSLATE(STRING, OUT, IN, PAD)
```

其中，STRING 为原字符串，IN 指原字符串中需要被替换的字符，OUT 指对应要替换成的新字符，这里的字符是一一对应的，如果在 OUT 中未指定对应的新字符，那么就用 PAD 所定义的字符来填充，默认用空格填充。

```
TRANSLATE('abbc','&','b') -> 'a&&c'    /* 用&来替换字符 b */
TRANSLATE('abcdef','12','abcd','.') -> '12..ef' /* 用..来替换 cd */
TRANSLATE('APQRV',,'PR') -> 'A Q V'   /* 用空格来填充替换字符 */
```

4.5.6　其他内置函数

本节介绍其他的内置函数，其中 EXTERNALS 和 USERID 是 TSO 提供的函数，见表 4-6。

表 4-6　其他内置函数汇总

函　　数	功能描述	示　　例
ADDRESS	返回当前命令提交执行的环境	ADDRESS() -> ' TSO'
ARG	返回传递给子例程的参数信息	ARG(1) ; ARG(2)
BITAND	返回两个字符串按位与运算的结果，并与较长的输入字符串长度相同	BITAND('73'x,'27 'x) ->'23'x
BITOR	返回两个字符串按位或运算的结果	BITOR('15'x,'24'x) ->'35'x
BITXOR	返回两个字符串按位异或的结果，逻辑运算可使用占位符	BITXOR('12'x,'22'x) ->'30'x
CONDITION	返回当前捕捉到的特殊情况的信息	CONDITION('C') -> 'FAILURE'
DATE	按照指定格式返回当前系统日期	DATE() -> ' 2 Oct 2011'
ERRORTEXT	根据出错编号返回错误信息的内容，在 0～99 之间，该错误编号是系统指定的，用于调试程序	ERRORTEXT(16) -> 'Label not found'
EXTERNALS	返回终端输入缓冲区的字符数	EXTERNALS() -> 0
LINESIZE	返回终端的行宽减 1	LINESIZE()　->79
QUEUED	返回外部数据队列中剩余的行数	QUEUED() -> 5

续表

函　数	功能描述	示　例
SOURCELINE	返回程序的行数或某一行的内容	SOURCELINE() -> 10 SOURCELINE(1) ->'/* REXX*/'
TIME	返回指定格式的系统时间	TIME() -> '16:54:22'
TRACE	返回跟踪设置或选择跟踪状态	TRACE() -> '?R' TRACE('O') -> '?R'
USERID	返回 TSO 下的用户 ID，非 TSO 下返回特定的 ID，作业步或作业名	USERID() -> 'TE01'
VALUE	返回变量值或给变量赋予新值	firstName="Jane" VALUE('firstName') -> 'Jane' K=3; fred='K'; VALUE(fred,5) -> '3' VALUE(K) -> 5
XRANGE	返回包含指定首尾元素的一位编码的字符串（EBCDIC、ASCII 等），根据具体系统选择不同编码方式	XRANGE('a','f ') -> 'abcdef ' XRANGE(,'04'x) -> '0001020304'x

下面重点介绍一些函数的用法。

1. ARG

ARG 函数在无参数的情况下，返回传递的参数的数量；如果只是指定了一个具体的数字，则返回对应位置的参数；如果指定了数字和一个选项参数，则检测对应位置的参数是否符合一定规则，符合返回 1，不符合返回 0，可选的参数如下（只需大写字母即可）。

- Exists：判断对应位置的参数是否存在。
- Omitted：判断对应位置的参数是否未指定。

```
/*REXX 中调用语句："Call name 'a',,'b';"*/
ARG()        -> 3
ARG(1,'E')   -> 1
ARG(3,'O')   -> 0
```

2. CONDITION

CONDITION 函数可得到捕捉事件的名字，描述信息，执行捕捉的命令或事件的状态，通过可选参数来标识需要返回的信息（C，D，I，S）。

```
CONDITION()         -> 'CALL'          /* 默认情况,同 I */
CONDITION('C') -> 'FAILURE'        /* Condition name -- 名称 */
CONDITION('I') -> 'CALL'           /* Instruction -- CALL 或 SIGNAL */
CONDITION('D') -> 'FailureTest'    /* Description -- 描述信息 */
CONDITION('S') -> 'OFF'            /*Status -- 状态,ON OFF DELAY*/
```

如在程序中设置了 CALL ON ERROR，则将会对 ERROR 事件进行跟踪，并且捕捉到一条出错语句，那么此时使用 CONDITION 函数的结果如下。

```
CONDITION()     ->   'SIGNAL'
CONDITION('C')  ->  'ERROR'
CONDITION('I')  ->  'CALL'
CONDITION('D')  ->  'PULL'  /* 出错语句 */
CONDITION('S')  ->  'ON'
```

如果没有设置捕捉事件，则 CONDITION 函数返回空值。如果有子例程中使用 CALL ON 等指令设置事件捕捉，则 CONDITION 函数是针对子例程中的设置进行记录的，在子例程返回后，CONDITION 将恢复到 CALL 命令执行之前的状态。

3. DATE

DATE 函数在默认情况下返回的日期格式是 dd mm yyyy，没有前导的 0 或空格。此外，可以进行日期格式的转换，输入一定格式的日期，并且不能有前导的 0 或空格，还要注意大小写形式，然后以一定格式输出。可选的格式如下。

- Base：返回从 0001 年 1 月 1 日到当前日期之间的天数，格式为 dddddd。可通过除 7 取余数得到当前日期是一周的第几天。
- Century：返回从 0001 年 1 月 1 日到当前日期之间的天数，以 100 天为单位计算。
- Days：返回当前日期是当前年份中的第几天，包括当天。
- European：返回日期格式为 dd/mm/yy。
- Julian：返回日期格式为 yyddd。
- Month：返回当前月份的完整英文表示，首字母大写，其余字母小写。
- Normal：返回日期格式为 dd mon yyyy，月份为对应单词的缩写形式。
- Ordered：返回日期格式为 yy/mm/dd，便于排序。
- Standard：返回日期格式为 yyyymmdd，便于排序。
- Usa：返回日期格式为 mm/dd/yy。
- Weekday：返回当前日期在一周中的对应的英文单词。

假设当前日期是 2001 年 11 月 20 号，对应示例如下。

```
DATE(,'20010709','S') -> '9 July 2001'
DATE('B','25 Sep 2001') -> '730752'
DATE('C') -> '690'
DATE('E') -> '20/11/01'
DATE('J') -> '01324'
DATE('M') -> 'November'
DATE('N','1438','C') -> '8 Dec 2003'
DATE('O') -> '01/11/20'
DATE('S') -> '20011120'
DATE('U','25 May 2001') -> '05/25/01'
DATE('W') -> 'Tuesday'
```

4. EXTERNALS

EXTERNALS 函数返回当前输入缓冲区中的字符数量，但是在 TSO/E 环境下，系统中没有对应的缓冲区，所以返回值总是为 0。

5. LINESIZE

LINESIZE 函数返回当前程序中一行内容纳的字符数量。如果 REXX 程序在 TSO/E 后台运行，即 JCL EXEC PGM=IKJEFT01，该函数总是返回 131；如果在前台运行，该函数返回当前终端的宽度。在非 TSO/E 环境下，LINESIZE 函数返回 OUTDD 文件（默认是 SYSTSPRT）中逻辑记录的长度。

6. TIME

TIME 函数返回系统时间可以有如下格式形式，并且返回的时间除 0 外都无前导 0 或空格。

- Civil：返回时间格式为 hh:mmxx，其中 xx 为 am 或 pm。
- Elapsed：返回时间格式为 sssssssss.uuuuuu，返回从设置的起始时间开始经过的时间。
- Hours：返回当前时间是一天中的第几个小时，格式是 hh 或 h。
- Long：返回时间格式为 hh:mm:ss.uuuuuu，uuuuuu 为毫秒。
- Minutes：返回当前时间是一天中的第几分钟，最多四个字母，格式为 mmmm。
- Normal：返回默认格式的时间：hh:mm:ss。
- Reset：返回时间格式为 sssssssss.uuuuuu，返回从 Elapsed 设置的时间开始经过的时间，并重置 Elapsed 的开始时间。
- Seconds：返回当前时间是一天中的第几秒，最多五位表示，格式为 ssssss。

假设当前时间是 2:50 a.m.，对应示例如下。

```
TIME()     -> '02:50:22'
TIME('C') -> '2:50am'
TIME('H') -> '02'
TIME('L') -> '02:50:22.123456'
TIME('M') -> '0170'      /* 50 + 60*2 */
TIME('N') -> '02:50:22'
TIME('S') -> '01042'     /* 22 + 60*(50+60*2) */
```

TIME 函数还可以计算从某一时刻开始经过的时间，方法是：用 TIME('E')或 TIME('R')设置起始时间，此时返回 0，并且此后再调用 TIME('E')或 TIME('R')函数将返回从上一次设置的起始时间开始经过的时间，以秒为单位，例如：

```
TIME('E') -> 0 /* 设置初始时间 */
/* 停顿一秒 */
TIME('E') -> 1.002345
/*停顿一秒*/
TIME('R') -> 2.004690
```

4.6　TSO/E 外部函数

除前面介绍的内置函数之外，TSO/E 还提供了许多具有特定功能的外部函数，有些函数的功能和 CLIST 中的控制变量相同。

可用的 TSO/E 外部函数有 GETMSG、LISTDSI、MSG、MVSVAR、OUTTRAP、PROMPT、SELANG、STORAGE、SYSCPUS、SYSDSN 和 SYSVAR。其中，MVSVAR、SETLANG、STORAGE 和 SYSCPUS 函数可以运行在任意地址空间，即 TSO/E 和非 TSO/E 环境，其余函数只能运行在 TSO/E 环境下的 REXX 程序中。下面将介绍这几个外部函数的使用方法和功能特性。

4.6.1 GETMSG 函数

使用 GETMSG 函数和 TSO/E 中的 CONSOLE 以及 CONSPROF 命令可以在 REXX 程序中对 MVS 进行操作。通过 CONSOLE 命令启动一个外部的 MVS 控制台会话，执行一些 MVS 命令，并且用 CONSPROF 命令来控制那些路由到用户控制台的消息不被显示，之后可以使用 GETMSG 函数来得到这些不被显示的消息，用户还能通过条件控制来搜索符合一定条件的消息和类型。相关的消息内容和消息信息存储在变量中，这些变量可以被 REXX 程序使用。

调用 GETMSG 函数，得到一个返回值，并且通过指定的变量存储消息的内容。函数每次返回一条消息，消息本身可以是多行的，每一行的消息顺序地存储在变量中。

函数的返回码如表 4-7 所示。

表 4-7 GETMSG 函数返回码

返回码	描　　述
0	GETMSG 函数执行成功，得到了系统消息
4	GETMSG 函数执行成功，但是没有得到系统消息，原因可能是： ● 没有消息需要读取 ● 消息和函数中设置的搜索条件不符 ● 在函数中设置了读取时间，在时间界限内没有读取到消息
8	GETMSG 函数执行成功，但是在函数执行时按了中断键，导致未读取消息
12	GETMSG 函数执行不成功，未启动控制台会话，需要先用 CONSOLE 命令
16	GETMSG 函数执行不成功，在其执行期间，控制台会话被终止

函数的调用形式及各变量的功能如下。

```
GETMSG(msgstem, msgtype, cart, mask, time)
```

msgtem 设置 GETMSG 函数放置返回消息的变量的词干，可以是一个变量序列或者是一个复合变量，消息会从前到后依次放入对应变量中。如有 3 行消息返回，假设指定复合变量“msg.”，接收，那么 3 行消息将依次放入“msg.1”、“msg.2”和“msg.3”中，“msg.0”中存放变量的总数；如果指定变量序列 msg 接收，那么 3 行消息将依次放入 msg1、msg2 和 msg3 中，msg0 中存放变量的总数。

msgtype　设置希望提取的消息类型，可以是以下任意一种。

- SOL（solicited）：返回请求消息，即 MVS 或子系统对用户命令的回应。
- UNSOL：返回非请求消息，可以是其他用户发送的消息或广播消息。

- EITHER：返回以上两种类型的消息，是默认值。

cart 和 mark 这两个参数用来筛选返回消息。其中 cart 参数设置发出 CONSOLE 命令的应用程序的 ID 号，通过与 mask 的值做与运算，得到要筛选的条件，GETMSG 函数通过这个值在路由到用户控制台的消息队列中搜索符合条件的结果并返回。只有在 msgtye 设置为 SOL 时 cart 和 mark 参数的设置才有效。cart 和 mark 可以是 1～8 位的字符或 1～16 位的十六进制字符串，如'C1D7D7C1F4F9F4F1'X。如果位数不够，将在高位补 0，位数超过将截断低位字符，通常 MASK 的前 4 位是 FFFFFFFF，保证程序 ID 值不变。

time　设置 GETMSG 函数等待消息返回的时间，如果在设置的时间段内消息没有路由到用户的控制台，函数将返回，默认情况下 time 的值为 0。

下面列举了一些常见的 GETMSG 函数。

```
MSG = GETMSG('CONSMSG.','SOL')
MCODE = GETMSG('DISPMSG.','SOL',,,120)
MSGRETT = GETMSG('DMSG','SOL','C1D7D7D3F2F9F6F8'X,,60)
CONMESS = GETMSG('MSGC.','SOL','APPL ','FFFFFFFF00000000'X,30)
```

4.6.2　LISTDSI 函数

LISTDSI（List Dataset Information）函数可以得到数据集的详细属性信息，这些信息存储在指定的变量中，可以用在指令中，通过返回码确定函数是否执行成功。可以直接用数据集名称或存储该名称的变量来传递请求，但是如果在其他卷上有同名的数据集或这个数据集已经被打开，则不能使用 LISTDSI 函数。

LISTDSI 函数支持访问 DASD 上的数据集和使用绝对世代名称的 GDG 数据集，不支持磁带和 HFS 数据集。通常，该函数用来确定数据集的特征信息或者用来创建一个具有相同属性的新的数据集。如果访问的数据集是 VSAM 类型，则只返回卷 ID，设备号和数据集的组织形式信息；如果是访问存储在多个卷上的数据集，函数只返回第一个卷的信息；如果访问的是一个文件，并且含有多个数据集，函数只返回第一个数据集的信息。

函数的返回码如表 4-8 所示。

表 4-8　LISTDSI 函数返回码

返回码	描　　述
0	函数执行成功，得到数据集信息
4	数据集中有些信息（通常是目录信息）未获得，其他返回信息可认为是有效的
16	函数失败，未得到任何属性信息

函数的调用形式和参数信息如下。

```
LISTDSI(dataset-name, location,(file-name), directory, recall,smsinfo)
```

- dataset-name：设置要查询的数据集名称。
- location：设置要查询的数据集的位置，可以指定 VOLUME 和 PREALLOC，如果未指定这两个参数就在系统目录中搜索。
- filename：如果想指定文件名，则需要加 FILE 关键字，然后指定文件的名称。
- directory：设置是否要返回 PDS 数据集的目录信息，DIRECTORY 或 NODIRECTORY，默认是不返回，这样可以提高执行效率。
- recall：设置是否要查询被 DFHSM 转移的数据集，RECALL 或 NORECALL，如果指定 RECALL，那么会搜索所有转移的数据集，如果指定 NORECALL，而该数据集又被转移了，则会报错。如果未指定 RECALL 或 NORECALL，则系统只搜索转移到 DASD 上的数据集。
- smsinfo：设置是否要返回 SMS 相关信息，包括数据集类型、使用空间、数据类名称、存储类名称和管理类名称。使用 SMSINFO 或 NOSMSINFO，只使用与被 SMS 管理的数据集。

函数返回后，将数据集的信息存储在对应变量中，表 4-9 列举几项常用的变量。

表 4-9　LISTDSI 函数的返回信息

变　量	存 储 内 容
SYSDSNAME	数据集的名称
SYSVOLUME	盘卷 ID 号
SYSUNIT	盘卷所在的设备号
SYSDSORG	数据集的组织形式：PS，PSU，DA，DAU，IS，ISU，PO，POU，VS
SYSRECFM	记录的格式：U，F，V，T，B，S，A，M，???（Unknown）
SYSLRECL	逻辑记录的长度
SYSBLKSIZE	数据集的块大小
SYSUSED	数据集分配时的空间单位，不适用于 PDSE
SYSKEYLEN	VSAM 数据集中键值的长度
SYSCREATE	数据集的创建时间，格式为 year/day，如 2011/135
SYSDSSMS	数据集的类型信息，为 SMS DSNTYPE 中的信息，如果未读取到，返回的信息：SEQ、PDS、PDSE 如果是 PDSE 数据集，那么将返回 LIBRARY、PROGRAM_LIBRARY、DATA_LIBRARY
SYSRACFA	返回数据集的 PROFILE 信息：NONE、GENERIC、DISCRETE
SYSREASON	LISTDSI 的返回码

下面给出了一些 LISTDSI 的示例。

```
X = LISTDSI('MYREXX.EXEC')
X = LISTDSI("'SYS1.PROJ.NEW'")

/* REXX */
X = LISTDSI(WORK.EXEC)
```

```
SAY 'FUNCTION CODE FROM LISTDSI IS: ' X    /* 输出 0 */
SAY 'THE DATA SET NAME IS: ' SYSDSNAME     /* 输出 USERID.WORK.EXEC */
SAY 'THE DEVICE UNIT ON WHICH THE VOLUME RESIDES IS:' SYSUNIT
/* 输出 3390 */
SAY 'THE RECORD FORMAT IS: ' SYSRECFM      /* 输出 FB */
SAY 'THE LOGICAL RECORD LENGTH IS: ' SYSLRECL  /* 输出 80 */
SAY 'THE BLOCK SIZE IS: ' SYSBLKSIZE       /* 输出 800 */
SAY 'TYPE OF RACF PROTECTION IS: 'SYSRACFA /* 输出 GENERIC */
```

4.6.3　MSG 函数

MSG 函数返回 TSO/E 消息显示的状态——ON 或 OFF，即是否显示 TSO/E 消息。指定 ON 或 OFF 参数，设置当前 TSO/E 的消息是否显示，同时函数返回之前的消息状态。

```
MSGSTAT = MSG() -> 'OFF'    /* 返回系统当前设置为“OFF” */
STAT = MSG('OFF') -> 'ON'   /* 返回之前设置“ON”,并关闭信息显示*/
```

如果在 REXX 程序中指定 MSG(OFF)，则在程序或其子函数运行期间，消息是不显示的，直到使用 MSG(ON)命令，或原程序执行结束。如果从原程序中用 EXEC 调用其他的 CLIST 程序，对消息显示的设置不会传递到 CLIST 程序中，默认情况下，消息会在新程序中显示。MSG 函数和 CLIST 中的 CONTROL MSG 和 CONSTROL NOMSG 函数作用相同。下面是一个实例，用于确保在使用 TRANSMIT 命令后输出消息内容。

```
IF MSG() = 'OFF' THEN,
"TRANSMIT node.userid DA(myrexx.exec)"
ELSE
DO
x = MSG("OFF")
"TRANSMIT node.userid DA(myrexx.exec)"
a = MSG(x) /* 输出消息内容 */
END
```

4.6.4　MVSVAR 函数

MVSVAR 函数返回 MVS，TSO/E 和当前会话的信息，如 MVS 系统的代号名称，TSO/E 会话的安全级别等。通过给函数传递参数，设置要返回的信息内容，参数如表 4-10 所示。

表 4-10　MVSVAR 函数的参数信息

参 数 值	描　　述
SYSAPPCLU	APPC/MVS 的逻辑单元（LU）的名字
SYSDFP	MVS/DFP（Data Facility Product）的级别
SYSMVS	z/OS 系统的组件 BCP（Base Control Program）的级别

续表

参 数 值	描 述
SYSNAME	REXX 程序运行的系统的名称，是 SYS1.PARMLIB 中 IEASYSxx 成员的 SYSNAME 参数的值
SYSSECLAB	TSO/E 会话的安全级别
SYSSMFID	正在使用 SMF（System Management Facility）管理的系统 ID
SYSSMS	判断 DFSMS/MVS 是否可用的标识：UNAVAILABLE、INACTIVE、ACTIVE
SYSCLONE	标识 MVS 系统名称的参数
SYSPLEX	MVS Sysplex 的名称，在 COUPLExx 或 LOADxx 中
SYMDEF	MVS 系统的系统参数

```
/*REXX*/
IF MVSVAR('SYSDFP') >= '00.03.03.00' THEN  /* 判断是否可以使用 SYSSMS 参数*/
SAY MVSVAR('SYSSMS')                       /* 可能输出为 ACTIVE */
EXIT 0
```

通常，MVSVAR 函数用于获取 MVS 系统信息，以便在使用某些功能之前判断其可用性等，防止错误地使用不合乎规范的程序等。具体的参数信息请参见 IBM 白皮书 *TSO/E REXX Reference*。

4.6.5 OUTTRAP 函数

OUTTRAP 函数将命令的输出放在一组变量中，REXX 程序可以处理这些变量中的输出内容，调用格式如下。

```
OUTTRAP((off),varname,max,concat)
```

- off：关闭错误跟踪。
- varname：可以是复合变量的词干或者数字变量的前缀，复合变量可以用索引来读取变量，但是变量序列则不能用索引来循环读取。
- max：设置捕捉结果的最大行数，可以设置一个具体的数值、*或者空格。如果指定*或者空，则所有的输出都将被捕捉。默认情况下，最多可以捕捉 999 999 999 行，超过的部分将被舍弃。
- concat：设置输出的方式，可以是 CONCAT 或 NOCONCAT。
  - CONCAT 设置命令被顺序地捕捉直到设定的最大值，如一条命令有 3 行输出，那么它们分别存在以 1、2、3 结尾的变量中，下一条命令产生两条输出，存储在以 4、5 为结尾的变量中，是默认设置。
  - NOCONCAT 设置每次命令的输出都从变量 1 开始存储，即如果第一条命令有 3 行输出，存在变量 1、2、3 中，第二条命令有两条输出，那么将覆盖第一条命令的输出，从变量 1 开始存储。

```
x = OUTTRAP("ABC",4,"CONCAT")
```

```
/* 假设命令有 3 行输出，则 ABC 变量的结果如下 */
ABC0 --> 3
ABC1 --> 第一行输出的内容
ABC2 --> 第二行输出的内容
ABC3 --> 第三行输出的内容

/* 如果此时又有一条命令有两行输出，那么第二行输出将不被跟踪 */
ABC4 --> 第二条命令的第一行输出  /* ABC0 到 ABC3 的值不变 */
```

REXX 程序可以通过变量来显示和处理 TSO/E 命令的输出结果，其中 TSO/E 命令的错误信息会被捕捉，但是其他类型的错误信息会直接被发送到终端。捕捉从发起该命令的语句开始，包括它调用的子句，直到原程序结束或者是捕捉程序被关闭。

OUTTRAP 也可以捕捉在 REXX 程序中发出的命令的输出，但是写在 REXX 程序中的命令不能停掉捕捉的过程，需要由源程序操作捕捉流程。OUTTRAP 捕捉 TSO 命令输出示例如下。

```
/* REXX Source Code */
call outtrap 'line.'         /* or x=outtrap('line.') */
"listds te02.rexx.lab"      /* IDCAMS command         */
call outtrap 'off'          /* or y=outtrap('off')    */
say "lines : "line.0
do i=1 to line.0
say line.i
end

/* 运行结果 */
lines : 5
TE02.REXX.LAB
--RECFM-LRECL-BLKSIZE-DSORG
  FB    80    32720   PO
--VOLUMES--
  USER02
```

4.6.6 PROMPT 函数

PROMPT 函数可以设置是否开启或关闭与 TSO/E 的交互，或者返回当前关于提示信息的设置。当设置 PROMPT 为 ON 状态时，REXX 程序可以发出 TSO/E 命令提醒用户未指定的参数值等。

```
x = PROMPT('ON')/* x 中保存了原 PROMPT 的设置，并且开启了信息提醒 */
```

如果只是想知道当前关于 PROMPT 的设置，则可以不指定任何参数。

```
x = PROMPT()
```

如果数据栈非空，命令会先从数据栈中取数据显示，可以用 NEWSTACK 命令来防止错误提示。如果 TSO/E 的 PROFILE 中指定了 NOPROMPT，那么即使调用 PROMPT（ON）函数也无法显示提示信息，可以通过 PROFILE 命令来控制在 TSO/E 会话中交互信息的使用。

4.6.7 SETLANG 函数

SETLANG 函数返回一个 3 位字符码，表示当前 REXX 程序所显示的语言种类。也可以通过指定参数来设置当前需要显示的语言类型。可行的字符码如表 4-11 所示。

表 4-11 SETLANG 的参数设置表

字符码	语言种类	字符码	语言种类
CHS	简体中文	ENU	英语（大小写）
CHT	繁体中文	ESP	西班牙语
DAN	丹麦语	FRA	法语
DEU	德语	JPN	日语
ENP	全大写英语	KOR	韩语
PTB	葡萄牙语		

默认情况下，REXX 显示的信息的类别与是系统中的语言相同，同时指定的语言类型需要先在系统中安装过才能使用，不然会报错。

```
oldlang = SETLANG(ENU) /* oldlang 中存储了原本的语言类型设置 */
/* 之后的显示信息都将是英文 */
```

4.6.8 STORAGE 函数

STORAGE 函数可以从系统中指定地址处取得数据或者放置数据，具体用法如下。

```
STORAGE(address,length,data)
```

其中，address 参数是一个十六进制数，代表了要读取数据的地址，length 指定了要读取的字符长度，默认是一位，data 参数用来设置要写入该地址的数据，与 lengh 设置的长度无关。如果 STORAGE 函数读取的地址越界了，那么只对地址范围内的空间进行操作。

```
storret = STORAGE(000AAE35,25) /* 从 000AAE35 处读取 25 个字符 */
storrep = STORAGE(0035D41F,,'TSO/E REXX')
/* 将 0035D41F 处的数据替换成'TSO/E REXX' */
```

4.6.9 SYSCPUS 函数

SYSCPUS 函数在一个复合变量中存储当前在使用的 CPU 的信息，其中 CPU 的数量

放在 stem.0 变量中，其余的变量中依次存储了 CPU 的序列号。同时，SYSCPUS 函数的返回值为 0 或 4，0 表示执行成功，4 表示执行成功，但是在执行的过程中检测了到一些不一致的情况，可能是在用的 CPU 数量在执行时发生了变化。

```
/* REXX */
X = SYSCPUS('CPU.')
SAY '0, IF FUNCTION PERFORMED OKAY: ' X
SAY 'NUMBER OF ON-LINE CPUS IS ' CPU.0
DO I = 1 TO CPU.0
SAY 'CPU' I ' HAS CPU INFO ' CPU.I
END

/* 执行结果如下 */
0, IF FUNCTION PERFORMED OKAY: 0
NUMBER OF ON-LINE CPUS IS 2
CPU 1 HAS CPU INFO FF0000149221 /* 后 4 位信息是模型号 */
CPU 2 HAS CPU INFO FF1000149221 /* 前 6 位信息是 CPU 的 ID 号 */
```

4.6.10　SYSDSN 函数

SYSDSN 函数判断指定的数据集名称是否存在或该数据集是否可用，指定的数据集可以是顺序数据集或分区数据集成员，另外，如果指定的是分区数据集的成员，则系统会检查对该成员是否有访问的权限。

通过 SYSDSN 函数可以获得 DASD 上的数据集信息，对于磁带并不支持，而世代数据集（GDG），支持绝对世代名称的查询。可以使用如下方法指定数据集的名称。

```
X = SYSDSN("'USERID.PRO.REXX'")     /* 避免在数据集前加前缀 */
X = SYSDSN('MYREXX.EXEC')           /* 系统会自动增加前缀查找 */
VAR = "EXEC"
X = SYSDSN(VAR)                     /* 查找名称'USERID.EXEC' */
Y = SYSDSN('VAR')                   /* 查找名称'USERID.VAR' */
```

查询数据集的状态信息，如果数据集可用，则返回 OK，如果数据集不可用或者不存在，将返回以下信息的一种。

```
MEMBER SPECIFIED, BUT DATASET IS NOT PARTITIONED
MEMBER NOT FOUND
DATASET NOT FOUND
ERROR PROCESSING REQUESTED DATASET
PROTECTED DATASET
VOLUME NOT ON SYSTEM
UNAVAILABLE DATASET
INVALID DATASET NAME, data-set-name:
MISSING DATASET NAME
```

下例中，通过判断数据集的可用性来决定调用的子程序。

```
X = SYSDSN("PRO.REXX(MEM)")
IF X = 'OK' THEN
CALL ROUTINE1
ELSE
CALL ROUTINE2
```

4.6.11 SYSVAR 函数

SYSVAR 函数返回 MVS、TSO/E 和当前会话的信息，如软件安装等级、用户登录过程、用户 ID，等等，通过具体的参数设置返回信息的内容，可以是以下类型。

1. 用户信息

- SYSPREF：用户 PROFILE 的前缀，通常是用户 ID。
- SYSPROC：当前会话的登录过程名，如果是批处理作业，返回 INIT。
- SYSUID：当前会话中登录的用户的 ID，与 USERID 函数返回的结果相同。

2. 终端信息

- SYSLTERM：返回当前终端可以显示的行数，后台运行时返回 0。
- SYSWTERM：返回显示的屏宽，后台运行时返回 132。

3. 语言信息

- SYSDTERM：返回当前终端是否支持显示 DBCS 字符。
- SYSKTERM：返回当前终端是否支持显示 Katakana 字符。
- SYSPLANG：返回用户 PROFILE 表（UPT）中记录的主要使用语言。
- SYSSLANG：返回用户 PROFILE 表（UPT）中记录的次要使用语言。

4. 程序信息

- SYSENV：返回当前程序运行是在前台（FORE）还是后台（BACK）。
- SYSICMD：返回隐式调用当前程序的命令或程序的名称，显式调用则返回空。
- SYSISPF：返回当前程序是否可用 ISPF 对话服务（ACTIVE/NOT ACTIVE）。
- SYSNEST：判断当前的程序是否被其他 REXX 或 CLIST 调用。
- SYSPCMD：返回最近执行的命令的名称。
- SYSSCMD：返回最近执行的子命令的名称。

5. 系统信息

- SYSCPU：返回当前会话中 CPU 使用的时间，以秒为单位。
- SYSHSM：返回 DFHSM 的安装级别。
- SYSJES：返回 JES 系统的名称和等级。
- SYSLRACF：返回 RACF 的等级。
- SYSNODE：返回 JES 系统的网络结点名称。

- SYSRACF：返回当前 RACF 是否可用，取值 AVAILABLE，NOT AVAILABLE，NOT INSTALLED，分别对应了安装且可用，安装但不可用和未安装的状态。
- SYSSRV：返回当前会话中使用的 SRM 服务的单元数。
- SYSTERMID：返回 REXX 程序执行的终端 ID。
- SYSTSOE：返回 TSO/E 的等级。

6. 控制台会话信息

- SOLDISP：判断命令返回的请求信息在路由到用户控制台时是否显示在终端上。
- UNSDISP：判断非请求信息在路由到用户控制台后是否显示在终端上。
- SOLNUM：返回可以存储的请求信息的数量。
- UNSNUM：返回可以存储的非请求信息的数量。
- MFTIME：判断用户是否设置要在系统信息中显示时间戳。
- MFOSNM：判断用户是否设置要在系统信息中显示系统名称。
- MFJOB：判断用户是否设置要在系统信息中显示作业名或作业 ID。
- MFSNMJBX：判断用户是否设置在系统信息中不显示系统名称和作业名称。

```
SAY SYSVAR("SYSJES")    /* 可能输出 JES2 OS 2.10 */
SAY SYSVAR("SYSRACF")   /* 可能输出 AVAILABLE */
```

SYSVAR 函数可以用来在使用一些命令或者调用一些数据集之前查看他们的相关信息，并根据结果进行适当的选择，同时，用户也可以设置作业的执行、信息的显示等，来定制自己的系统。

4.6.12　函数包

函数包是一组外部函数或子例程打包在一起，可以更快速地调用。它们不是写在 REXX 程序中然后被逐个翻译执行的，而是要首先生成中间代码，然后链接得到可执行模块并被调用。一些符合条件的语言有 COBOL、汇编、PL/I 等。

函数包可以分为以下三种。

（1）用户自定义：用户自定义的外部函数，在调用时最先被搜索，通常是用来重载其他函数包的。

（2）本地函数包：一般只对一组用户使用的函数，在用户函数包之后被搜索。

（3）系统函数包：是可以在整个系统范围内使用的函数包，如 TSO/E 外部函数，最后被搜索。

用户自定义的函数需要在登录时被载入，默认的用户函数包名称是 IRXFUSER，默认的本地函数包是 IRXFLOC，其他函数包可以在系统参数中指定名称。

使用自定义函数包时，首先用户需要自己编写需要引用的函数或例程，然后为函数表编写一个目录，指定函数包头部信息和每个函数的入口地址，然后将这些外部函数和目录编译链接成一个可执行模块。具体请参见 IBM 白皮书 *TSO/E REXX Reference*。

在了解了 REXX 函数和子例程的组成后，总结一下调用函数时，系统对该函数的搜

索顺序。

（1）内部函数，写在同一个 REXX 程序中的函数。

（2）内置函数，搜索 REXX 或 TSO/E 内置的函数。

（3）函数包，按照用户自定义、本地和系统函数包的顺序搜索。

（4）模块库，搜索系统中的可执行模块。

（5）外部函数，在程序所在的分区数据集或系统库 SYSEXEC 和 SYSPROC 中搜索。

第 5 章

REXX 数据处理

5.1 数据解析

REXX 中丰富的数据解析功能是该语言的一大特色。REXX 中的解析（parsing）主要是将数据分解为更小部分（通常是指字符串）到一个或者多个变量之中。数据解析同样也可用于数据格式化的操作。

5.1.1 常用解析指令

常用到的有如下几条解析指令：

- PULL 指令；
- ARG 指令；
- PARSE VAR 指令；
- PARSE VALUE-WITH 指令。

1. PULL 指令

PULL 指令在前面已经提到过，被用作为从终端交互式地读入变量值。

```
say ' please enter 2 word'  /*用户输入"hello world" */
PULL word1 word2            /*word1 取得值 HELLO,word2 取得值 WORLD*/
```

事实上，PULL 还可用于取得数据栈中的信息。PULL 将字符信息读入后，转为大写信息并赋值给一个或多个变量。

PARSE PULL 指令和 PULL 指令类似，取得对应位置的变量值，所不同的是字符读入后，不做大写转换直接将原始信息赋给对应变量，因此可以将 PULL 命令可以看作是 PARSE UPPER PULL 命令的缩略形式。

2. ARG 指令

ARG 指令用于取得传递给 REXX 程序（exec）、函数以及子例程的参数值，并将放入到对应的一个或者多个的变量值中。注意 ARG 和 PULL 一样，在将参数值赋给对应变量时，会将参数值转换成大小字母。例如要执行一个名称为 USERID.REXX.EXEC(ARGEX)，可以使用 EXEC 指令在命令行传递参数。

```
EXEC REXX(ARGEX) 'hello world.' Exec
```

若 ARGEX 中使用 ARG word1、word2 接受参数的传递，word1 中的值将赋值为 HELLO，word2 值为 WORLD。同样，PARSE ARG 在将参数值赋予给对应变量时，不会对其进行大写转换。指令 ARG 也就是指令 PARSE UPPER ARG 的缩略形式。

3. PARSE VAR 指令

PARSE VAR 指令将某个变量值分解到该变量之后的一个或多个变量之中。PARSE VAR 之间没有 UPPER 关键字，因此变量解析时不对其进行大小写转换。

```
sentence ='Rexx language is good'
PARSE VAR sentence string1 string2 string3
SAY string1   /*  string1 变量值为 Rexx        */
SAY string2   /*  string2 变量值为 lauguage   */
SAY string3   /*  string3 变量值为 is good    */
```

4. PARSE VERSION 指令

PARSE VERSION 指令返回 REXX 的版本号和 REXX 处理器发布的时间。

```
PARSE VERSION ver
SAY ver   /*  ver 变量值为 REXX370 3.48 01 May 1992   */
```

5. PARSE VALUE-WITH 指令

PARSE VALUE [expression] WITH 指令是将某个表达式，根据 WITH 关键字，解析为一个或多个变量，在没有 UPPER 关键字时，在变量解析时不对其进行大小写转换，其功能大体和 PARSE VAR 相似，例如上例中的 PARSE VAR 语句可以直接替换为 PARSE VALUE [expression]WITH 指令。但该指令比 PARSE VAR 更灵活的一点在于：VAR 所指定的源数据(字符)串只能是变量(例如上例中的 sentence)，而 VALUE [expression] WITH 中指代的源数据（字符）串不仅可以是变量（例如 sentence），也可以是直接的字符串数据。

```
PARSE VALUE 'Rexx is good' WITH var1 var2 var3 var4
                  /* var1 取得"Rexx" ; var2 取得"is" */
                  /* var3 取得"good" ; var4 为空('') */
```

由上述叙述可知，可以将以上四条指令都统一为 PARSE 关键字指令。事实上，PARSE 指令具有如下形式。

```
PARSE [UPPER] valueToParse [template_list]
```

这里 valueToParse 可以是 PULL、ARG、SOURCE、VAR variableName、VERSION、VALUE [expression] WITH 等操作数之一。

这里 template_list 指定了数据解析的变量模板或者叫做变量格式。template_list 经常指定了一个变量模板，例如前面例子中的“var1 var2 var3”就是一个变量模板；也可以通过逗号分隔指定多个模板；也可以省略。正是通过了 template_list 的指定，REXX 得

以完成形式丰富的各种数据解析工作。

5.1.2 template_list 详解

数据解析是通过对比源数据和模板（或变量模型）把源数据解析成单独的数据。REXX 数据解析通常分两步：

① 将源数据（字符串）解析为模板格式指定的子串。

② 将各子串放入对应位置的变量中，称为分解而成的单词（word）。

在 template_list 省略，即不指定对应的变量模板时，REXX 依然进行源数据的“解析”准备，但却不进行后续的变量赋值。例如，执行 template_list 默认的指令 PARSE PULL，效果就是将数据栈中当前数据串删除；执行默认 template_list 的指令 PARSE VALUE [expression] WITH 能简单完成表达式 expression 的计算。

本节主要介绍的是 template_list 为单一变量模板的情况。变量模板指定了源数据（字符串）解析的匹配模式，它使用分隔符分割的变量名列表，分隔符可以为空格、占位符、字符分隔符或者位置分隔符。

1. 空格分隔符

最简单的变量模板就是一组用空格分隔的直接变量名列表：

```
variable1 variable2 variable3 ….
```

例如，上述示例中的“var1 var2 var3 var4”就是这种变量模板，作用是将源串通过空格分隔解析为对应单词，每个变量（除去最后一个变量）都取得一个相应单词。如果源串所包含的单词数多于列表的变量数，则变量列表的最后一个变量名将取得数据串经过解析后的所有剩余部分，即最后一个变量可能包含了多个单词，并且还可以含有空格。

```
PARSE VALUE 'Value with Blanks.' WITH pattern type
                           /* pattern 取得 "Value"      */
                           /* type 取得 "  with Blanks."*/
```

如果源串包含的单词数少于列表的变量数，则最后变量被置为空值（"），如下面所示。

```
PARSE  VALUE  'Hello World.' WITH data1 data2 data3
                           /* data1 取得 "Hello"        */
                           /* data2 取得 "World."       */
                           /* data3 取得 ""              */
```

2. 占位符分隔符

除了使用变量获取单词内容，也可以使用句点作为占位符，句点（.）占位符可看作是一个空变量（dummy variable）使用，即解析出的单词不放入实际的变量中；或用作收集不需要的串尾信息使用。

```
PARSE VALUE 'Value with Periods in it.' WITH pattern . type .
```

```
    /*句点所对应的'with'以及'in it'两个单词,不放入实际变量中*/
SAY pattern                       /* pattern 为"Value" */
SAY type                          /* type 为"Periods"  */
```

3. 字符（变量）分隔符

当需要将源数据（字符串）以其中源串中特定字符或者标点进行分隔解析时，可以使用字符（变量）作为解析分隔符，解析分隔符指定了以何为标志进行源串分解，分解之后的单词中不再含有该分隔符。例如：

```
phrase = 'To be, or not to be?'    /*希望以源字串中的逗号分隔单词  */
PARSE VAR phrase part1 ',' part2   /*此处，用','表示该处分隔符     */
SAY part1                          /*part1 ='To be'                */
SAY part2                          /*part2 =' or not to be?'       */
```

注意，上例中逗号不再包含在前后的单词之中，因为逗号被作为分隔符号使用。进一步地，REXX 允许将分隔符通过带括号的变量表示，从而增强了编程的灵活性。下例中，使用变量 separator 表示分隔符“,”，输出结果相同。

```
separator = ','
phrase = 'To be, or not to be?'
PARSE VAR phrase part1 (separator) part2
SAY part1                     /*part1 ='To be'            */
SAY part2                     /*part2 =' or not to be?'   */
```

4. 位置分隔符

REXX 的变量模板中还允许使用数字标定列的方式指定源数据（字符串）的分隔解析方式。通常用无符号整数指定串的绝对位置，而用带符号的整数指定其相对位置。

（1）列的绝对位置

一个无符号的整数或者用“=”连接的整数，标志了变量模板中变量开始的绝对位置。第一个字段从第一列开始，但不包含随后标志列绝对位置数字所在的行。随后字段开始在接下来的位置，即列数指定的列的位置。

```
quote = 'Ignorance is bliss.'
PARSE VAR quote part1 5 part2
SAY part1           /*  part1  取得 "Igno"              */
SAY part2           /*  part2  取得 "rance is bliss."   */
```

这个例子也可以改写成下面的例子。可以看出下例中特定位置指示符“1”可以推断出，上例中默认起始位置为第一列，下例中，“=5”指示出 part2 的起始绝对位置为第 5 列。

```
quote = 'Ignorance is bliss.'
PARSE VAR quote 1 part1 =5 part2
SAY part1           /*  part1  取得 "Igno"            */
SAY part2           /*  part2  取得 "rance is bliss."   */
```

当一句解析语句中的位置分隔符多于一个，并且后面的数值小于前面的数值，则将后面变量的数据解析位置重新设置为第一列，从头开始计数进行循环。

```
quote = 'Ignorance is bliss.'
PARSE VAR quote part1 5 part2 10 part3 1 part4
SAY part1       /* part1 取得 "Igno"                */
SAY part2       /* part2 取得 "rance"               */
SAY part3       /* part3 取得 " is bliss."          */
SAY part4       /* part4 取得 "Ignorance is bliss."  */
```

当解析语句中的变量在其前面和后面都有位置分隔符时，这两个分隔符表示此变量对应数据的起始和结束位置。

```
quote = 'Ignorance is bliss.'
PARSE VAR quote 1 part1 10 11 part2 13 14 part3 19 1 part4 20
SAY part1       /* part1 取得 "Ignorance"           */
SAY part2       /* part2 取得 "is"                  */
SAY part3       /* part3 取得 "bliss"               */
SAY part4       /* part4 取得 "Ignorance is bliss."  */
```

（2）列的相对位置

在解析数据时，在模板中有符号的整数标志了列的相对位置。也就是说，一个部分的起始位置由前一个位置分隔符决定。有符号的整数可以是正的（+）或者负的（-），表示了此部分的起始位置从上一个部分的起始位置右移（用“+”）或左移（用“-”）。第一个部分的起始位置分隔符省略不写，默认为 1。在下面的例子中，“=5 part2”表示，第二部分的起始位置为行：1+5=6；同样地，第三部分的起始位置为行：6+5=11，依此类推。

```
quote = 'Ignorance is bliss.'
PARSE VAR quote part1 +5 part2 +5 part3 +5 part4
SAY part1               /* part1 contains "Ignor"       */
SAY part2               /* part2 contains "ance "       */
SAY part3               /* part3 contains "is bl"       */
SAY part4               /* part4 contains "iss."        */
```

减号的使用与加号类似，减号用来在数据字符串中回退（左移）。在下例中，默认的 part1 从第一列开始；“+10 part2”：part2 从第 1+10=11 列开始；“+3 part3”：part3 从第 11+3=14 列开始；接下来“-3 part4”：part4 从第 14-3=11 列开始。

```
quote = 'Ignorance is bliss.'
PARSE VAR quote part1 +10 part2 +3 part3 -3 part4
SAY part1               /* part1 contains "Ignorance "  */
SAY part2               /* part2 contains "is "         */
SAY part3               /* part3 contains "bliss."      */
SAY part4               /* part4 contains "is bliss."   */
```

5.2 数据栈操作

同其他常用语言一样，REXX 支持对栈的操作。在 REXX 中的栈称为数据栈，数据栈与普通意义上的栈存在着一些差别。本节介绍如何使用 REXX 中的数据栈来存储信息。

5.2.1 什么是数据栈

TSO/E REXX 的使用一个可扩展的数据结构来存储信息，此结构称为数据栈。数据栈结合了传统栈和队列的特点。

栈和队列是类似的数据结构类型，用于临时存放数据项（元素）。当元素被使用时，它们就会从数据结构的顶部被删除。栈和队列的根本区别是在添加元素时(见图 5-1)。在栈中增加元素时增加到顶部，而队列中则增加到底部。

使用堆栈时，最后一个添加到栈的元素（elem6）是第一个被移除的。因为对于堆栈，增加和删除的对象是放在栈顶部的元素，这种技术被称作后进先出（Last In First Out，LIFO）。

使用队列时，第一个添加到队列的元素（elem1）是第一个被移除的。因为元素从队列的底部增加并从队列的顶部删除。这种技术被称作先进先出（First In First Out，FIFO）。

如图 5-2 所示，在添加元素时，REXX 中的数据栈使用了栈和队列的综合技术。元素可以从数据栈的顶部和底部增加。但移除元素时，则只能从顶部移除。

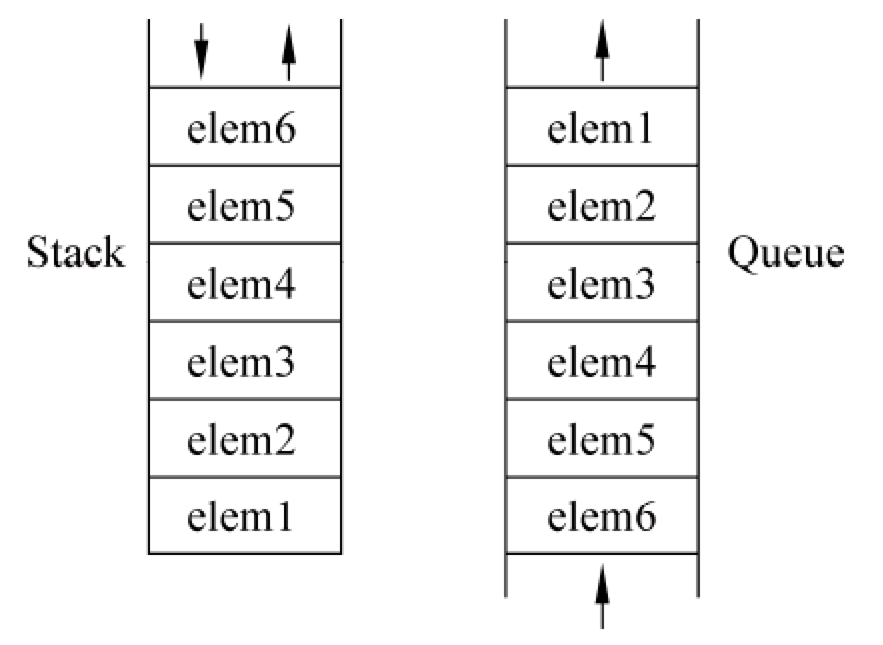


图 5-1　栈和队列结构示意图

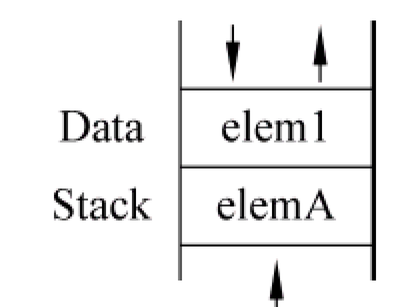


图 5-2　数据栈结构示意图

5.2.2 数据栈操作指令

REXX 中有几个指令用来操作数据栈，其中两个指令用来往数据栈中添加元素，另一个指令用来从数据栈中移除元素。

1. 添加数据栈元素

在数据栈中存储信息数据有两条指令：PUSH 和 QUEUE。

（1）PUSH：在数据栈的顶部增加数据。对增加的数据项长度没有限制。

```
elem1 = 'String 1 for the data stack'
PUSH elem1
```

（2）QUEUE：在数据栈的底部增加数据。对增加的数据项长度没有限制。

```
elemA = 'String A for the data stack '
QUEUE elemA
```

如果上述两个指令在同一个 exec 中对同一个数据栈进行操作，情况如图 5-3 所示。

2. 删除数据栈元素

从数据栈中移除数据使用 PULL 和 PARSE。

前面提到 PULL 指令用于从终端获取数据（当数据栈为空时，PULL 指令为从终端获取数据作用）。确切地说，PULL 和 PARSE 用于从数据栈顶端移除数据。

```
PULL stackitem
SAY stackitem       /*  displays STRING 1 FOR THE DATA STACK  */
```

使用上述指令，变量 stackitem 包含了 elem1 元素转化为大写的值。

如果在 PULL 指令前增加 PARSE 指令，得到的数据就不会被转换成大写。

用上面两条指令操作完数据栈后，数据栈如图 5-4 所示。



图 5-3　添加数据栈元素操作示意 1

图 5-4　删除数据栈元素操作示意 2

3. 数据栈元素数量

内置函数 QUEUED 可以返回一个数据栈中的元素数量。例如，为了得到一个数据栈中的元素数目，可以无参数地使用 QUEUED 函数。

```
SAY QUEUED()            /*  displays a decimal number  */
```

使用 QUEUED 函数，从一个数据栈中移除所有元素并显示它们，例如：

```
number = QUEUED()
DO number
   PULL element
   SAY element
END
```

5.3　文 件 读 写

本节描述如何利用 REXX 实现对数据集的 I/O 处理。在 TSO/E REXX 语言中的 EXECIO 命令能够实现对数据集的输入输出（I/O）操作。这些信息可以被存储在数据栈

中进行依次的处理，或者存储在变量列表中进行任意顺序的处理。

5.3.1 什么时候使用 EXECIO 命令

在 EXECIO 命令中存在很多的操作数，允许实现很多种类的 I/O 操作。例如，用户可以使用 EXECIO 命令来实现如下操作。

- 从一个数据集中读取数据。
- 向一个数据集中写入数据。
- 打开一个数据集，并不进行读写操作。
- 清空一个数据集。
- 从一个数据集中拷贝数据到另一个数据集中。
- 从一个复合变量列表中读取数据存入数据集中。
- 向一个顺序数据集末尾增加数据。
- 一次更新数据集中的一行数据。

5.3.2 EXECIO 命令简介

EXECIO 是 TSO/E REXX 命令。它的格式如下。

```
EXECIO Lines/*  DISKR/DISKRU/DISKW   DDName   LineNum   Read/WriteParms
```

其中：

- Lines/*：代表处理的记录数，*代表整个数据集。
- DISKR/DISKRU/DISKW：DISKR 表示以只读方式处理文件，DISKRU 表示读取记录允许更新文件，DISKW 表示写文件。
- DDName：代表文件，可以通过 JCL 的 DD 语句来关联 DDName 和文件，也可以通过 Allocate 语句在 REXX 程序中关联 DDName 和文件。
- LineNum：开始读的起始记录，只适用于 DISKR 或 DISKRU，对于 DISKW 没有此参数。
- Read/WriteParms：一些读写参数，如 STEM、OPEN 和 FINIS 等，在 5.3.4 节会详细介绍。

EXECIO 命令从数据集中读取信息可以使用 DISKR 或者 DISKRU 操作数。使用这两个操作数，可以实现打开数据集，但不读取数据集的任何记录。EXECIO 命令向数据集存储信息使用 DISKW 操作数，使用这个操作数，同样可以打开数据集但不读取数据集的任何记录，而且可以实现清空数据集。

在使用 REXX 中的 EXECIO 命令在读取或写入数据到数据集之前，数据集一定要满足下面的要求。

（1）是顺序数据集或是 PDS 的一个成员。

（2）与要写入的内容有相符合的数据集特性。

使用 EXECIO 可以从一个数据集读取数据，然后放到数据栈中，存储在数据栈中的数据可以存储为 FIFO 形式或 LIFO 性质。FIFO 形式是默认的。也使用 EXECIO 从一个数据集读取数据，放入到一个复合变量列表 var 中，数据集中的第一行数据存储在 var.1 中，第二行数据存储在 var.2 中，依此类推。在数据被存入数据栈和变量列表以后，REXX 可以使用这些数据，或者将它们复制到另一个数据集中，再或者更新这些数据然后重新存入原始的数据集中。

使用 EXECIO 中的 DISKR 或者 DISDRU 操作数，可以从一个数据集中读取信息，放到数据栈或者一个变量列表中。使用 DISKW 操作数可以向数据集写入信息。

更多内容请参阅 IBM 白皮书 z/OS TSO/E REXX Reference。

5.3.3　文件读取

根据使用输入数据集的用途，可以使用 DISKR 或者 DISKRU 参数。

（1）DISKR：对数据集进行只读操作，可以使用带有 FINIS 选项的 DISKR 参数来初始化 I/O。FINIS 选项在读取信息之后关闭数据集。假如想要继续读取数据集中的下一行，则不要在 EXECIO 语句中使用 FINIS 选项。

```
"EXECIO   *   DISKR  myindd  (FINIS"
```

（2）DISKRU：对数据集进行读取更新操作，数据集内容可以读到数据栈中或者放到复合变量中。对读取的最后一条记录可以进行更新操作，然后通过 DISKW 命令将更新后的记录写回数据集中。

```
"EXECIO   1   DISKRU  myindd  (OPEN"
"EXECIO   1   DISKW   myindd  (FINIS"
```

1. DISKR 和 DISKRU 参数选项

可以使用下面的 DISKR 和 DISKRU 参数选项。

- OPEN：打开数据集。当在 EXECIO 0 后面指定 OPEN 参数，则打开数据集，并且将打开位置的指针放于第一条记录之前。

```
"EXECIO  0  DISKR  myindd  (OPEN"
```

- FINIS：读完数据集后关闭数据集。关闭后，允许其他 REXX 程序通过相应的 DDNAME 使用此数据集。同时，将当前位置指针指到数据集的起始位置。
- STEM：读取信息到一个可被索引的复合变量中，或者一个有数字标识的变量列表。指定 STEM 参数后跟一个变量名，来保证该变量列表（而不是数据栈）将接收文件的内容。

```
"EXECIO   *   DISKR   myindd   (STEM  newvar."
```

在上例中，复合变量是 newvar，从数据集中得到的信息被存储在变量 newvar.1、newvar.2、newvar.3 等中。复合变量中的元素数目将存储在一个特殊变量 newvar.0 中。

假如 REXX 读取 10 行数据到 newvar 参数中，则 newvar.0=10。接下来的变量 newvar.1 包含第一条记录，newvar.2 包含第二条记录，依次到 newvar.10 包含第十条记录。之后的变量（比如 newvar.11、newvar.12.）包含了之前 EXECIO 命令输入的值。

为了避免混乱，错把之前的输入值作为当前的变量值，可以用 EXECIO 命令清除整个变量列表的值。为了清除所有的变量项，可以使用 DROP 指令。

```
DROP  newvar.
```

或者将所有变量值设为零。

```
newvar.  = ''
```

- SKIP：跳过数据集中的特定行，不把他们输入数据栈或变量中。

```
"EXECIO  24  DISKR  myindd  (SKIP"
```

- LIFO：以 LIFO 的顺序读取栈中的信息。也就是说，REXX 将使用 PUSH 指令把数据集信息放到数据栈中。
- FIFO：以 FIFO 的顺序读取栈中的信息。也就是说，REXX 将使用 QUEUE 指令把数据集信息放到数据栈中。假如不指定 LIFO 还是 FIFO，FIFO 是默认的。
- 输入数据集：一个 I/O 数据集必须首先定义成 DDNAME 才能通过 EXECIO 进行操作。为了不与系统中之前已经存在的 DDNAME 冲突，REXX 对文件操作之前一般会定义一个新的 DDNAME，比如 myindd。例如：

```
"ALLOC  DA(my.in.data)  F(myindd)  SHR  REUSE"
"EXECIO  *  DISKR  myindd  (FINIS"
```

2. 文件读取示例

示例：想要打开一个数据集但不读取任何记录，则在 EXECIO 命令中选用 OPEN，指定 0 行。

```
"EXECIO  0  DISKR  myindd  (OPEN"
```

示例：想要从第 100 行打开一个数据集，并不读取任何行到数据栈中。

```
"EXECIO  0  DISKR  myindd  100  (OPEN"
```

示例：读取特定的行，在 EXECIO 命令中选用 OPEN 参数，并指定行数。

```
"EXECIO  25  DISKR  myindd  (OPEN"
```

示例：读取完整的数据集，在 EXICIO 命令中选用 OPEN 参数，并指定行数为*。

```
"EXECIO  * DISKR  myindd  (OPEN"
```

示例：想要从第 100 行开始读取 5 行到数据栈中，然后关闭数据集。

```
"EXECIO  5  DISKR  myindd  100  (FINIS"
```

示例：想要从数据集中读取所有记录，然后关闭数据集。

```
"EXECIO  *  DISKR   myindd  (FINIS"
```

示例：不想从数据集的起始位置开始读取，而是指定行数来开始读取。例如，从第 100 行开始读取数据并存到数据栈中。

```
"EXECIO   *   DISKR   myindd   100   (FINIS"
```

5.3.4　文件写入

使用 DISKW 操作数可以从数据栈或者复合变量列表中向数据集中写入信息。一个典型的将所有行写到数据集的 EXECIO 命令如下。

```
"EXECIO * DISKW myoutdd (FINIS"
```

1. DISKW 参数选项

可以使用下面的 DISKW 参数选项。

（1）OPEN

可以使用 EXECIO　0 命令来打开一个数据集，这个命令将文件位置指针置于文件的第一项记录之前。如果文件已经是打开的，该命令不执行任何操作。

```
"EXECIO 0 DISKW myoutdd (OPEN"
```

（2）FINIS

操作后关闭数据集。执行这个命令以后，其他的 execs 可以访问该数据集和它的 ddname。当 FINIS 被指定的时候，所有数据栈中的数据会被写到数据集中，从而强制完成 I/O 操作。

```
"EXECIO * DISKW myoutdd (FINIS"
```

（3）STEM

在 STEM 关键字后指定复合变量或者变量列表， REXX 将要写的源数据是这些变量，而不是数据栈。

```
"EXECIO * DISKW myoutdd (STEM newvar."
```

在这个示例中，STEM 后面指定了变量 newvar，这个复合变量中的所有数据将被写入数据集中，这些数据包括 newvar.1、newwar.2、newvar.3 等。

写数据时，变量 newvar.0 实际上并没有使用，当指定了* 的时候，EXECIO 命令会向数据集中写数据直到发现了一个没有初始化的变量。在这个例子中，如果复合变量包含了 10 个元素，EXECIO 命令将在 newvar.11 时停止。EXECIO 命令也可以指定从一个复合变量列表中写数据的行数。

```
"EXECIO 5 DISKW myoutdd (STEM newvar."
```

在这个示例中，EXECIO 命令从 newvar 中写 5 项数据，即使其中可能包含未初始化的变量。

（4）输出数据集

同文件读取类似，输出数据集必须首先分配给一个 ddname，实际上，分配一个新的 ddname 如 myoutdd 是一种更好的方法，可以在 exec 中直接使用 ALLOCATE 命令分配新的 ddname，如下所示，也可以在 exec 执行前先分配好。

```
"ALLOC DA(my.out.data) F(myoutdd) OLD REUSE"
"EXECIO * DISKW myoutdd ..."
```

2. 文件写入示例

示例：想在文件中写入 25 行数据，在 EXECIO 命令后直接指定要写的行数。

```
"EXECIO 25 DISKW myoutdd (FINIS"
```

示例：想在文件中写整个数据栈，或者直到发现一个空行为止，可以在 EXECIO 后面接星号，当 EXECIO 后面接星号的时候，这条命令将持续地从数据栈中取出数据，直到找到一个空行。如果到栈空时，仍然没有发现空行，EXECIO 将提示在终端输入，直到用户输入了一个空行。所以如果用户不想有终端 I/O 发生，请在栈底用一个空行来指示数据的结尾。

```
"EXECIO * DISKW myoutdd (FINIS"
```

示例：想要写入全部数据，但如果数据本身包含有空行，可以使用内置的 QUEUED() 函数来指示数据栈中有多少项。

```
n = QUEUED()
"EXECIO" n "DISKW outdd (FINIS"
```

示例：想清空数据集，可以先用 OPEN 选项打开一个数据集，而不向其中写记录。REXX 将文件位置指针置于第一条记录之前；然后使用 FINIS 选项关闭数据集，REXX 将向数据集中写入一个 end-of-file 标记，这样就可以清空数据集。注意数据集的状态属性一定不能使用 MOD，否则数据集内容将不能被清除。

```
"EXECIO 0 DISKW myoutdd ... (OPEN"
"EXECIO 0 DISKW myoutdd ... (FINIS"
```

示例：从顺序数据集 TE02.TEST1 复制内容，写入分区数据集 TE02.TEST2(MEM1) 中。

```
/* REXX */
"ALLOC DA(TE02.TEST1) F(myindd) SHR REUSE"
"ALLOC DA(TE02.TEST2(MEM1)) F(myoutdd) OLD"
"NEWSTACK"
"EXECIO * DISKR myindd (FINIS"       /* 读文件      */
QUEUE ''                             /* 标识文件结束  */
```

```
"EXECIO * DISKW myoutdd (FINIS"    /* 写文件          */
"DELSTACK"
"FREE F(myindd myoutdd)"
```

示例：更改顺序数据集 TE02.EMPLOYEE.LIST 的第二条记录。

```
"ALLOC DA('TE02.EMPLOYEE.LIST') F(UPDATEDD) OLD"
"EXECIO 1 DISKRU UPDATEDD 2"
PULL line
PUSH 'Crandall, Amy AMY 5500'
"EXECIO 1 DISKW UPDATEDD (FINIS"
"FREE F(UPDATEDD)"
```

5.3.5　EXECIO 的返回码

当一个 EXECIO 命令执行完，就会有一个 REXX 的特殊变量 RC 作为返回码返回。有效的 EXECIO 返回码在表 5-1 中列出。

表 5-1　EXECIO 返回码含义

返回码	意　　义
0	请求的操作正常结束
1	在做 DISKW 操作时，数据被删减
2	在执行 DISKR 或者 DISKRU 操作时，在没有读到程序中指定的行时，文件就读完了。(此返回码在用“*”指定行数时则不会发生)
4	在执行 DISKR 或者 DISKRU 操作连接数据集时，发现数据集为空。文件打开不成功且没有数据返回
20	严重错误。EXECIO 命令没有成功执行，返回错误信息

第 6 章

REXX 与子系统的交互

本章主要讨论将 REXX 用作一种主机系统应用架构（System Application Architecture，SAA）的过程化脚本语言。SAA 是指可用于设计和开发的一组软件接口、约定或协议的框架，这里 SAA 环境特指如下几个操作系统。

（1）MVS，包括 TSO/E，CICS，IMS。

（2）VM CMS。

（3）Operating System/400 (OS/400)。

（4）Operating System/2 (OS/2)。

SAA 过程化脚本语言被定义为 REXX 语言的一个子集，其目的是定义一个能在多种不同环境中使用的公共语言集。而 TSO/E REXX 就是在 MVS 系统上 SAA 的过程语言实现。本章将主要以 TSO/E REXX 为例，重点介绍相关的 REXX 高级议题。

6.1 执行宿主命令

REXX 命令调用是指非赋值、非关键字指令的调用，大致分为 REXX 命令和宿主命令（host command）两种。在 REXX 程序执行完每条命令后，特殊变量 RC 将被重置为当前命令执行后的返回码。因此通过对 RC 变量的查询，可判断程序执行过程中命令执行的正确与否。REXX 程序中可以执行很多不同类型和环境下的命令。

1. REXX 命令

REXX 命令是指 REXX 语言实现本身（这里指 TSO/E 实现下的 TSO/E REXX）所提供的完成相关程序操作的命令：包括了基本 I/O 控制、数据栈服务以及控制流机制（如前所述的交互式调试功能）等。关于 REXX 命令的详细介绍，参见第 3 章的内容。

2. 宿主命令

宿主命令则是指 REXX 程序执行的宿主环境所能识别的命令。所谓宿主环境，是指识别并执行特定命令的运行环境。REXX 程序中可以执行各种不同类型的宿主命令。

通常在程序执行前，需要在 REXX 程序代码中指定处理命令的宿主环境；在执行过程中，REXX 语言处理器一旦遇到了命令调用语句，就将该命令语句交给预先指定的宿主环境进行处理。在 REXX 程序中，如果将调用的命令加上引号，则可以直接执行。

下例为在 REXX 程序中调用 TSO/E 命令 ALLOC 的程序片段。该示例的作用是分配一个名为 NEW.DATA 的数据集，而该数据集的属性从数据集 OLD.DATA 继承。

```
"ALLOC DA(NEW.DATA) LIKE(OLD.DATA) NEW"
```

每个 REXX 程序至少有一个默认的宿主环境，如上例中，TSO/E 就是默认的宿主环境。如之前所述，在该宿主环境（TSO）下的 REXX 程序既可以执行 REXX 本身的命令（例如 I/O 控制信息的 EXECIO 命令），也可以执行 TSO/E 命令（例如分配数据集的 ALLOCATE 命令）。

TSO/E REXX 可用的宿主环境包括以下环境。

（1）TSO：在此宿主环境下，可在 TSO/E 地址空间中执行 TSO/E 命令和 TSO/E REXX 命令。

（2）MVS：在此宿主环境下，可在非 TSO/E 地址空间中执行 TSO/E REXX 命令。

（3）LINK：在此宿主环境下，可以链接相同级别的模块。

- LINKMVS：该宿主环境比 LINK 更进一步。可向被调用模块传递多个参数，且被调用模块能够修改这些参数。要注意的是，传递给被调用模块的参数中，包括了参数的长度标识。
- LINKPGM：该宿主环境与 LINKMVS 的区别在于，传递给被调用模块的参数中，不包括参数的长度标识。

（4）ATTACH：在此宿主环境下，可以将不同级别的模块链接到一起。

- ATTACHMVS：参见 LINKMVS。
- ATTACHPGM：参见 LINKPGM。

（5）ISPEXEC：在此宿主环境下，可以执行 ISPF 命令。

在 ISPEXEC 环境下，提供了近百个 REXX 程序（无论是在线程序还是批处理程序）可用的 ISPF 服务，包括对于交互式应用的相关支持，如显示面板、消息框、弹出窗口、菜单栏以及帮助等；对于操作数据集及数据集成员的支持（其功能类似于 ISPF 的 3.2、3.3 及 3.4 数据集操作选项）；对于数据的读写能力（可以替代 EXECIO 命令和 SAY & PULL 指令的作用）。ISPEXEC 环境还提供了调用 ISPF 表的功能，从而可以把结构化数据以行或列的形式（即是表的形式）存储。最后，ISPEXEC 环境提供文件裁剪的能力，该功能可以很好地帮助生成格式化的记录并存入数据集，而这些记录通常被用作 JCL 作业或系统实用程序（utilities）的数据输入。

（6）ISREDIT：在此宿主环境下，可以执行 ISPF/PDF EDIT 命令。

ISREDIT 环境提供的服务包括查找、修改、插入、删除数据以及搜索并修改编辑器中所有的环境变量等。因此，在 ISREDIT 环境下，开发者可以顺利地构建出类似 ISPF 下的数据集编辑面板，并通过进一步的命令调用来扩展该 Editor 面板的功能。

（7）CONSOLE：在此宿主环境下，可以执行 MVS 系统及子系统命令。

CONSOLE 环境下的命令被解析为 MVS 控制台命令，或者是子系统命令（如 JES2

命令），而对于这些命令的使用可以帮助开发者建立一些自动化操作的过程。关于 MVS 命令的详细使用信息，可以查看 IBM 白皮书 *MVS System Commands*。

（8）其他：除上述宿主环境外，还有 COMMCPI、LU62 以及 APPCMVS 等宿主环境。

值得注意的是，在不同的模式下，TSO/E REXX 提供的宿主环境并不相同。在 ISPF 模式下，以上所有的宿主环境都可以被使用（TSO 为默认宿主环境）。在 TSO/E READY 模式下，ISPEXEC 和 ISPEDIT 不能使用（TSO 为默认宿主环境）。而在非 TSO/E 环境下，除了 ISPEXEC 和 ISPEDIT，TSO 环境也不能使用，此时的默认宿主环境为 MVS。

指定或切换当前命令执行的宿主环境，都要通过 ADDRESS 指令，该指令的调用方式有如下两种形式：

```
Address EnvironmentName
Address EnvironmentName CommandExpression
```

前一种形式将所有后续命令处理的宿主环境都变为指定环境；而后一种形式仅仅改变的是 CommandExpression 参数所指定命令的宿主环境，该指令后的命令处理不受影响。

注意下例的程序片段，其中的 ADDRESS 指令只将第一条命令切换到 ISPF 宿主环境中执行，而之后的命令仍然在 TSO 宿主环境中执行。

```
ADDRESS  ISPEXEC  "EDIT DATASET("dsname")"
/*只有命令"EDIT DATASET("dsname")" 在 ISPF 宿主环境中执行*/
"ALLOC DA("dsname") F(SYSEXEC) SHR REUSE"
/*后面语句恢复到默认的 TSO 宿主环境*/
```

如果需要查看当前宿主环境，可以调用内置函数 CurEnvName= ADDRESS()，通过其返回值 CurEnvName 确定当前所处的宿主环境。

此外，若要 TSO/E REXX 中检测某宿主环境是否可用，可通过 SUBCOM 命令，例如：

```
ARG dsname
SUBCOM ISPEXEC
IF RC=0 THEN
ADDRESS ISPEXEC "SELECT PGM(ISREDIT)"
ELSE
    "EDIT" dsname
```

该例中，SUBCOM 命令检查了指定的宿主环境 ISPEXEC 是否存在，并将检查结果作为返回值存入 RC 变量中。

实际上，SUBCOM 命令查找宿主环境表（Host Command Environment Table），该表中存放所有可被使用的宿主环境的名称（如 TSO、MVS 等）。通过添加、删除、修改宿主环境表中的记录，用户可以对当前系统可用的宿主环境进行定制。

6.2　REXX 与 TSO 环境的交互

TSO/E 宿主环境是最常用的宿主环境，在该环境下，可以使用 REXX 十分便捷地调用 TSO/E 命令，编写脚本来进行系统的日常维护工作。

在 TSO/E 宿主环境中，REXX 可以直接调用 RACF TSO 命令，以帮助系统管理员进行大批量的操作，如账号分配、资源授权等工作。

下面的示例为组 GROUP 批量添加了 USER ID 为 USR001～USR040 的新用户，并为这些用户数据集分别创建了 RACF PROFILE 进行读写保护。

```
DO I = 1 TO 9
   'AU USR000'I' DFLTGROUP(GROUP) OWNER(GROUP) PASSWORD(pass)'
   'ADDSD USR000'I'.** GENERIC OWNER(GROUP) UACC(NONE)'
END
DO I = 10 TO 40
   'AU USR00'I' DFLTGROUP(GROUP) OWNER(GROUP) PASSWORD(pass)'
   'ADDSD USR00'I'.** GENERIC OWNER(GROUP) UACC(NONE)'
END
```

如果想要为 DB2 用户授予数据库权限，同样可以在 TSO/E 宿主环境下，调用 TSO/E 命令及 REXX 命令，批量生成 DCL（Data Control Language）语句，随后通过 Batch 的方式来执行。也就是说，用户可以编写 REXX 脚本来生成一个 JCL 作业，然后将 DCL 语句作为该 JCL 作业的输入，最后执行此作业完成对于 DB2 用户的批量授权。

具体的代码示例如下。

```
/* REXX */
/* allocate file GRTGEN, associating with a specific data set */
"ALLOC DA('IBMUSER.JCL(GRTGEN)') FILE(GRTGEN) SHR REUSE"
/* compound variables contain JCL statements */
JCL.1 = '//GRTNEW JOB NOTIFY=&SYSUID'
JCL.2 = '//JOBLIB   DD DSN=DSN910.SDSNLOAD, DISP=SHR'
JCL.3 = '//DDLTABLE EXEC PGM=IKJEFT01, DYNAMNBR=20, REGION=32M'
JCL.4 = '//SYSTSPRT DD SYSOUT=*'
JCL.5 = '//SYSPRINT DD SYSOUT=*'
JCL.6 = '//SYSUDUMP DD SYSOUT=*'
JCL.7 = '//SYSTSIN DD *'
JCL.8 = 'DSN SYSTEM(DB9G)'
JCL.9 = "RUN PROGRAM(DSNTEP2)  PLAN(DSNTEP91) -"
JCL.10 = " LIB('DSN910.DB9G.RUNLIB.LOAD')"
JCL.11 = "//SYSIN    DD  *"
/* compound variables contain SQL statements */
DO I = 1 TO 9
 PACKAGEADM.I = 'GRANT PACKADM ON COLLECTION IBM000'I' TO IBM000'I';'
 CREATEDBA.I = 'GRANT CREATEDBA TO IBM000'I';'
 BINDADD.I = 'GRANT BINDADD TO IBM000'I';'
END
```

```
DO I = 10 TO 40
 PACKAGEADM.I = 'GRANT PACKADM ON COLLECTION IBM00'I'STO IBM00'I';'
 CREATEDBA.I = 'GRANT CREATEDBA TO IBM00'I';'
 BINDADD.I = 'GRANT BINDADD TO IBM00'I';'
END
/* write all statements from compound variables into the file */
"EXECIO 0 DISKW GRTGEN (OPEN"
"EXECIO * DISKW GRTGEN (STEM JCL."
"EXECIO * DISKW GRTGEN (STEM PACKAGEADM."
"EXECIO * DISKW GRTGEN (STEM CREATEDBA."
"EXECIO * DISKW GRTGEN (STEM BINDADD."
"EXECIO * DISKW GRTGEN (FINIS"
/* free the file allocation */
"FREE FILE(GRTGEN)"
```

在该例中，REXX 首先调用 TSO/E 命令 ALLOCATE 分配一个名为 GRTGEN 的文件，而该文件实际关联到已有的数据集 IBMUSER.JCL(GRTGEN)。随后，将一些固定的 JCL 作业语句写入复合变量 JCL，其中包括指定 PGM = IKJEFT01，并在 SYSIN 中启动数据库 DSN 环境，运行 DSNTEP2 程序，该程序可以以批处理的方式来执行随后输入 SQL 语句。

然后以同样的方式将所有 SQL 授权语句写入复合变量 PACKAGEADM 以及 CREATEDBA 中之后，最后调用 TSO/E REXX 命令 EXECIO，该命令将之前几个复合变量的内容依次写入到事先分配好的文件 GRTGEN 中，也即数据集 IBMUSER.JCL(GRTGEN)之中。最后，再次使用 TSO/E 命令释放之前分配的文件 GRTGEN。

执行该 REXX 脚本后，查看 IBMUSER.JCL(GRTGEN)数据集，其内容如图 6-1 所示。

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
EDIT        IBMUSER.JCL(GRTGEN)   - 01.00                Columns 00001 00072
Command ===>                                                Scroll ===> CSR
****** ***************************Top of Data ******************************
000001 //GRTNEW JOB NOTIFY=&SYSUID
000002 //JOBLIB   DD   DSN DSN910.SDSNLOAD,DISP=SHR
000003 //DDLTABLE EXEC PGM=IKJEFT01,DYNAMNBR=20,TIME=1440,REGION=32M
000004 //SYSTSPRT DD SYSOUT=*
000005 //SYSPRINT DD SYSOUT=*
000006 //SYSUDUMP DD SYSOUT=*
000007 //SYSTSIN DD *
000008 DSN SYSTEM(DB9G)
000009 RUN PROGRAM(DSNTEP2)  PLAN(DSNTEP91) -
000010  LIB('DSN910.DB9G.RUNLIB.LOAD')
000011 //SYSIN     DD  *
000012 GRANT PACKADM ON COLLECTION IBM0001 TO IBM0001;
………
000091 GRANT CREATEDBA TO IBM0001;
………
000131 GRANT BINDADD TO IBM0001;
………

 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange    F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

图 6-1 IBMUSER.JCL(GRTGEN)内容

提交该作业，即可完成对于用户 IBM0001 到 IBM0040 的 DB2 相关授权。

6.3 REXX 与 MVS 控制台的交互

用户可以通过 TSO/E 的 CONSOLE 命令建立一个扩展的 MVS 控制台会话。通过激活该控制台会话，用户可以输入相应的 MVS 系统和子系统命令并获取相应的系统命令响应信息。用户在控制台会话中可使用的 MVS 系统及其子系统命令，取决于当前控制台用户的 MVS 命令授权。

通常，用户可以在 TSO/E READY 模式下，输入如下命令激活控制台会话。

```
CONSOLE ACTIVATE
```

相应地，通过 DEACTIVATE 关键字即可中止当前控制台会话。

```
CONSOLE DEACTIVATE
```

在激活控制台会话后，即可以在 REXX 程序中使用 SYSCMD 关键字使用 MVS 系统及其子系统命令。

```
"CONSOLE SYSCMD(system_command)"
```

正如之前所提到的，用户可通过 ADDRESS CONSOLE 命令将当前宿主环境切换为 CONSOLE，从而可以避免调用命令时对 SYSCMD 关键字的重复使用。下面的例子中展示了该模式。

```
/* REXX program ... */ ..
"CONSOLE ACTIVATE" ...
ADDRESS CONSOLE "mvs_cmd1" ..
"mvs_cmd2" ..
"mvs_cmd3" ...
"CONSOLE DEACTIVATE"
EXIT
```

在使用该控制台宿主环境之前，用户必须确保本身拥有控制台命令权限。如果当前控制台会话没有被激活，宿主环境 CONSOLE 将无法定位和识别 MVS 系统命令，RC 返回码被置为-3 表明当前有错误产生。

如果有关的控制台会话信息没有显示于终端，可通过 TSO/E 的外部功能函数 GETMSG 获取对应信息。CART（Command And Response Token）关键字可将 REXX 程序中执行的 MVS 系统命令以及 MVS 子系统命令同对应命令的响应输出关联起来，即给对应命令的相应输出一个标记值，可以用于检测该命令的当前响应输出是否和预期的响应输出一致。

下面的例子中，激活控制台会话后启动两台系统打印机（PRT1 和 PRT2），并为对应的每个 START 命令分别指定其唯一的 CART 值，并通过将 CART 值作为参数传递给检

测程序 CHKPRT。CHKPRT 对 CART 所对应的响应输出进行检测，并判断打印机是否启动成功，显示信息提示。

```
CONSPROF SOLDISP(NO) SOLNUM(400)
CONSOLE ACTIVATE
CONSOLE SYSCMD($S PRT1) CART('PRT10001')
CONSOLE SYSCMD($S PRT2) CART('PRT20002')
EXEC MY.EXEC(CHKPRT) 'PRT10001'
EXEC EXEC MY.EXEC(CHKPRT) 'PRT20002' EXEC
```

下面的例子是一个 CHKPRT 检测程序，通过 GETMSG 收集对应请求的响应输出，并置于复合词干 PRTMSG 中，PRTMSG.0 中为输出信息条数，PRTMSG.1 表示响应输出的第一条信息，通过将其与预期标志'$HASP000'进行匹配比较后，判断该命令执行情况，并打印出该控制命令执行的最终结果。

```
/* REXX exec to check start of printers */
ARG CARTVAL
 GETCODE = GETMSG('PRTMSG. ', 'SOL',CARTVAL,,60)
 IF GETCODE = 0 THEN
  DO
    IF POS('$HASP000',PRTMSG.1) = 0 THEN
    SAY "Printer started successfully."
    ELSE
          DO  INDXNUM = 1 TO PRTMSG.0
                                 /* PRTMSG.0 中为输出信息的行数*/
    SAY  PRTMSG.INDXNUM          /*输出信息全部显示*/
    END
END
ELSE
SAY "GETMSG error retrieving message.  Return code is" GETCODE
EXIT
```

6.4 REXX 与 JES 的交互

在通常情况下，REXX 通过基本的 TSO 命令即可以完成与 JES 的交互，如使用 TSO Submit 命令提交作业，使用 TSO Cancel 命令取消作业，使用 TSO Status 命令和 TSO Output 命令查看作业的执行信息等。

REXX 可以借助 TSO 的 Submit 命令、TSO/E 编辑器的 Submit 命令、控制台 Start 方式、EXECIO 内部读入器、FTP 等方式提交 JCL 作业。其中，最常使用的是 TSO 的 Submit 命令，在命令中指定 JCL 作业所在数据集，即可提交该作业。

一个最简单的通过 REXX 提交 JCL 作业的示例如下。

```
/* REXX */
```

```
"SUBMIT ('IBMUSER.REXX.JCL(EXECREXX)')"
```

执行该脚本，可以看到 IBMUSER.REXX.JCL(EXECREXX)作业已经被提交到 JES，并返回其执行情况。

```
JOB EXECREXX(JOB06802) SUBMITTED
***
12.11.14 JOB06802 $HASP165 EXECREXX ENDED AT SVSCJES2
MAXCC=0 CN(INTERNAL)
***
```

通过借助 REXX 强大的功能函数以及外部数据栈，可在 REXX 程序内部自由灵活地构建 JCL 作业，借此完成很多特定重复的作业编写，从而大量减少了重复的人工编码和提交工作。

通常可以在 REXX 中构建完成 JCL 作业代码后，通过将宿主环境转换为 TSO，并运行提交（Submit）命令，即可完成 JCL 作业的提交。例如，通过 ADDRESS TSO "SUBMIT '<DSN NAME>'"提交作业。

下例即为一个完整的通过 REXX 数据栈方式构建 JCL 作业，并通过 Submit 命令提交的实例。

```
/* REXX */
user = USERID()                    /* 获得当前调用者 ID     */
queue "//"user"A JOB ACCTR,NOTIFY="user",MSGCLASS=A "
queue "//REXX   EXEC PGM=IKJEFT01,                    "
queue "//        PARM='REXX1'                       "
queue "//SYSTSPRT DD SYSOUT=*                         "
queue "//SYSTSIN  DD DUMMY                           "
queue "//SYSPROC  DD DSN=SP005.REXX.SOURCE,DISP=SHR      "
queue "//                                           "
queue "$$                                           "
o = OUTTRAP("output.",,"CONCAT")
ADDRESS TSO "SUBMIT * END($$)"     /*调用 SUBMIT 命令提交作业 */
o= OUTTRAP(OFF)                    /* 提交后，显示命令响应输入*/
SAY output.2  /*输出 "IKJXXXXX JOB SP005A(JOBXXXX) SUBMITTED" */
RETURN
```

该 REXX 程序构建的 JCL 作业的功能为：通过实用程序 IKJEFT01 在批处理环境下后台调用另一个名为 REXX1 的 REXX 程序。在程序逻辑中，通过函数 USERID()取得当前调用者的用户 ID，并作为 REXX 变量 user 用在 NOTIFY="user"处，使得程序在每次调用时，根据情况自动决定信息返回时的用户 ID。

在本例中使用到了数据栈，所谓数据栈是指 TSO/E REXX 所使用的一种外部数据结构，该数据结构结合了栈（stack）和队列（queue）的特点，可以用于程序数据的存储与处理。这里使用的 queue 语句即是将最新构建的一条 JCL 语句加入到数据栈底部，而最早构建的语句被弹出到栈顶，即符合“先进先出”队列的特点。通过这种方法，可以在

数据栈中构建一个逻辑正确、结构完整的 JCL 作业，以方便后续的提交使用。

代码中所用的 OUTTRAP()函数的功能是将其前后所包围的 REXX 命令（这里是指 ADDRESS TSO "SUBMIT * END($$)"）所产生的屏幕输出，放入到复合词干“output.”中，其中 output.2 即可打印出 Submit 命令提交后通常产生的第二行屏幕响应，即提交完成的屏幕提示。由于指定了 NOTIFY="user"（当前用户 ID），因此随后的系统的响应以及返回码也会随之出现在终端屏幕上。

通过直接执行该 REXX 程序，可以实现在 REXX 程序内部提交作业，完成后台调用另一个 REXX 程序的功能。由于构建 JCL 的作业代码及其提交代码，都在程序内部自动完成，因而可以省去重复的手工编写及提交 JCL 作业的需要。

除了直接使用 REXX 自身的函数 OUTTRAP 来获取 TSO Submit 命令的作业执行结果外，另一种方法是使用 TSO 命令 TSO Status 以及 TSO Output。这种方法的优势在于用户有了更大的自由空间，他们可以选择获取需要的输出，并根据具体的参数值判断之后的操作；也可以将一个或多个作业执行结果输出保存到指定数据集中，以便于存档和之后的查阅。

如果要查看作业的执行结果，可以使用 TSO Status 命令，从 Spool 中取得关于该作业的状态信息，也即作业当前所在的队列（如某作业当前是否尚未执行，处于等待队列中；或者已经执行完毕，但还处于输出队列中）。获取某个特定作业状态的语法如下。

```
/* REXX */
"STATUS jobname(jobid)"
```

其中 jobname 为作业名，如果没有指定 jobid，则系统返回结果会包括所有相同作业名的作业集合。如果要同时查看多个作业的信息，则可以在 STATUS 命令后跟随多个 jobname 参数，参数之间以逗号间隔。STATUS 命令返回结果如下。

```
JOB IBM100S(TSU07457) EXECUTING
***
```

如果想要获得完整的结果，则可以执行 TSO Output 命令，该命令可以输出整个作业的全部信息。在实际环境下，用户通常并不需要打印作业的全部信息，而是根据自己的需求从中选取重要信息进行输出，如下面的示例中，利用 Output 命令，只输出了 Message Class 的信息。执行的作业如下。

```
//JWSD582 JOB 91435,MSGCLASS=X
// EXEC PGM=IEHPROGM
//SYSPRINT DD SYSOUT=Y
//DD2 DD UNIT=3330,VOL=SER=333000,
// DISP=OLD
//SYSIN DD *
   SCRATCH VTOC,VOL=3330=333000
/*
```

REXX 程序调用作业的部分输出如下。

```
/* REXX */
"Output JWSD582 Class(X)"
```

如果想要了解更多有关 TSO 命令的语法细节及其实例，可以参阅 IBM 白皮书 *TSO/E Command Reference* 以及 *TSO/E User's Guide*。如果想要进一步了解 REXX 如何在 JES 系统日常维护之中使用，则建议阅读随后关于 REXX 与 SDSF 的交互部分。

6.5　REXX 与 SDSF 的交互

SDSF 是主机系统管理不可或缺的工具之一，作为访问 JES 的子系统，从监测作业的当前状态、所在队列、具体输出（通过 Spool 访问），到执行 JES 命令启动或挂起某个任务或子系统，或者是查看系统日志，主机系统管理员都是使用 SDSF 来完成这些日常维护工作的。

IBM 从 z/OS V1R9 开始支持 REXX 语言的 SDSF 编程接口，基本上大部分以往需要人工交互的操作现在都可以通过 REXX 脚本完成，比如查看或修改指定作业的信息、查看或修改指定设备的信息，浏览以及打印 SYSOUT 数据集输出，等等。

REXX 语言的一大优势就是易于编写自动化脚本，把系统管理员从一些重复性的日常维护工作中解放出来。可以说，REXX 与 SDSF 的交互很好地结合了 SDSF 的强大功能以及 REXX 的简单易用，使主机工作人员受益匪浅。

要在 REXX 程序中执行 SDSF 命令，首先要切换到特定的宿主环境 SDSF 之下。当然，如果之前 SDSF 宿主环境不在系统宿主环境表之中，还需要将其加入宿主环境表内，如下所示。

```
rc = isfcalls('ON', SSTYPE=JES2)
```

值得注意的是，添加 SDSF 宿主环境并没有使用之前的 REXX SUBCOM 命令，而是调用了 SDSF 的 REXX 专属命令 ISFCALLS，而之后的 SSTYPE 参数则指定了 SDSF 对应操作的是 JES2 系统（如果不指定该参数，则由 SDSF 自行判断）。通常情况下，rc 的返回值有 5 种可能，除了 0 代表命令执行成功，返回值为 1 说明查询宿主环境失败，返回值为 2 说明添加宿主环境失败，返回值为 3 说明删除宿主环境失败，返回值为 4 则代表命令中有语法错误。在具体的操作上，SDSF 为 REXX 提供了两个专用命令，ISFEXEC 和 ISFACT。前者通常用于调用一个访问 SDSF 的面板或者执行 MVS 系统命令，而后者则用于执行具体的操作（action character）以及修改 SDSF 面板中特定列的属性。

SDSF 为 REXX 提供调用某特定操作面板的 API，而面板中的数据则以表格形式（即按照行和列格式）返回以便于用户操作，示意如图 6-2 所示。

因此，用户可以很方便地操作返回的词干变量（其格式与面板相对应），取出或修改特定的数据。例如，通过下面这条命令，REXX 调用了 SDSF 的状态（Status，ST）面板，并为其自动创建了对应的词干变量。

```
  Display  Filter  View  Print  Options  Search  Help
-------------------------------------------------------------------------------
SDSF STATUS DISPLAY ALL CLASSES                          LINE 1-35 (379)
COMMAND INPUT ===>                                              SCROLL ===> PAGE
PREFIX=*  DEST=(ALL)  OWNER=*  SYSNAME=
NP   JOBNAME  JobID    Owner    Prty Queue      C  Pos  SAff  ASys Status
     IBMUSER  TSU07436 IBM100S   15 EXECUTION           SOW1  SOW1
     SYSLOG   STC07406 +MASTER+  15 EXECUTION           SOW1  SOW1
     SDSF     STC07407 STRTASK   15 EXECUTION           SOW1  SOW1
     VTAM     STC07409 STRTASK   15 EXECUTION           SOW1  SOW1
     TCPIP    STC07410 TCPIP     15 EXECUTION           SOW1  SOW1
     JMON     STC07411 STCJMON   15 EXECUTION           SOW1  SOW1
     RSED     STC07413 STCRSE    15 EXECUTION           SOW1  SOW1
     INIT     STC07422 STRTASK   15 EXECUTION           SOW1  SOW1
     RACF     STC07426 STRTASK   15 EXECUTION           SOW1  SOW1
     TN3270   STC07427 TCPIP     15 EXECUTION           SOW1  SOW1
     EXITMVS  STC07429 STCOPER   15 EXECUTION           SOW1  SOW1

 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=IFIND     F6=BOOK
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

图 6-2 SDSF 面板

```
Address SDSF "ISFEXEC ST"
```

每列的数据都会存放同一个词干变量中，词干变量名并非图 6-2 之中的列名，而是用户预先在 ISFPARMS 中定义的名称。以图 6-2 为例，假定 JOBNAME 列所对应的词干变量名为 JNAME，则可以通过下面的语句来查询特定作业名的作业记录。

```
do ix=1 to JNAME.0 /* Loop for all rows returned */
if pos("RJONES",JNAME.ix) = 1 then
⋮
```

当查询到指定的作业后，可以通过 ISFACT 命令来对其进行操作或修改相关属性。ISFACT 操作需要包括三部分内容：首先是当前所在的面板，在这里指定的是 ST 面板；其次要给出作业所在位置的词干变量 TOKEN.number（number 的数值在本例中存放在变量 ix 之中）；最后则是定义了 ISFACT 命令具体动作的参数。示例如下。

```
Address SDSF "ISFACT ST TOKEN('"TOKEN.ix"') PARM(NP P)"
```

其中最后的 PARM（NP P）的含义是，指定作业所在行的 NP 栏数值被修改为 P，实际上其效果等同于图 6-3。

很容易发现，本例中 ISFACT 执行的操作实际上是删除名为 CICSTS41 的作业。最后在整个程序结束之前，需要再次调用 ISFCALLS 命令，从宿主环境表中移除 SDSF 环境变量。

```
rc=isfcalls('OFF')
```

通过 REXX 编写这样一个并不复杂的脚本，就可以免去系统管理员进入到 SDSF 的 Status 面板，人工寻找并删除作业的工作了，如果该操作步骤相当固定并且日常使用率

```
  Display  Filter  View  Print  Options  Search  Help
-------------------------------------------------------------------------------
SDSF STATUS DISPLAY ALL CLASSES                        LINE 1-35 (379)
COMMAND INPUT ===>                                            SCROLL ===> PAGE
PREFIX=*  DEST=(ALL)  OWNER=*  SYSNAME=
NP   JOBNAME  JobID    Owner    Prty Queue        C  Pos  SAff  ASys Status
     IBMUSER  TSU07436 IBM100S    15 EXECUTION            S0W1  S0W1
     SYSLOG   STC07406 +MASTER+   15 EXECUTION            S0W1  S0W1
     SDSF     STC07407 STRTASK    15 EXECUTION            S0W1  S0W1
     VTAM     STC07409 STRTASK    15 EXECUTION            S0W1  S0W1
     TCPIP    STC07410 TCPIP      15 EXECUTION            S0W1  S0W1
     JMON     STC07411 STCJMON    15 EXECUTION            S0W1  S0W1
     RSED     STC07413 STCRSE     15 EXECUTION            S0W1  S0W1
P    CICSTS41 STC07414 STCOPER    15 EXECUTION            S0W1  S0W1
     RACF     STC07426 STRTASK    15 EXECUTION            S0W1  S0W1
     TN3270   STC07427 TCPIP      15 EXECUTION            S0W1  S0W1
     EXITMVS  STC07429 STCOPER    15 EXECUTION            S0W1  S0W1
     TCAS     STC07430 STRTASK    15 EXECUTION            S0W1  S0W1

 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=IFIND     F6=BOOK
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

图 6-3　SDSF 面板操作

极高的话，则通过 REXX 与 SDSF 交互将任务进行自动化意义是显而易见的。

事实上，REXX 与 SDSF 交互能够进行的操作还远远不止这些，其他常用的命令与操作还有：使用 ISFLOG 命令访问系统日志，使用 ISFSLASH 命令执行 MVS 系统命令等。更详细的介绍及使用示例，可以参阅 IBM 白皮书 *SDSF Operation and Customization* 之中的有关章节：*Using SDSF with the REXX programming language*。

另外，如果需要语法上的详细指导，可以直接在 SDSF 的命令栏中输入 REXXHELP（缩写为 REXXH），以查看非常细致的在线帮助，如图 6-4 所示。

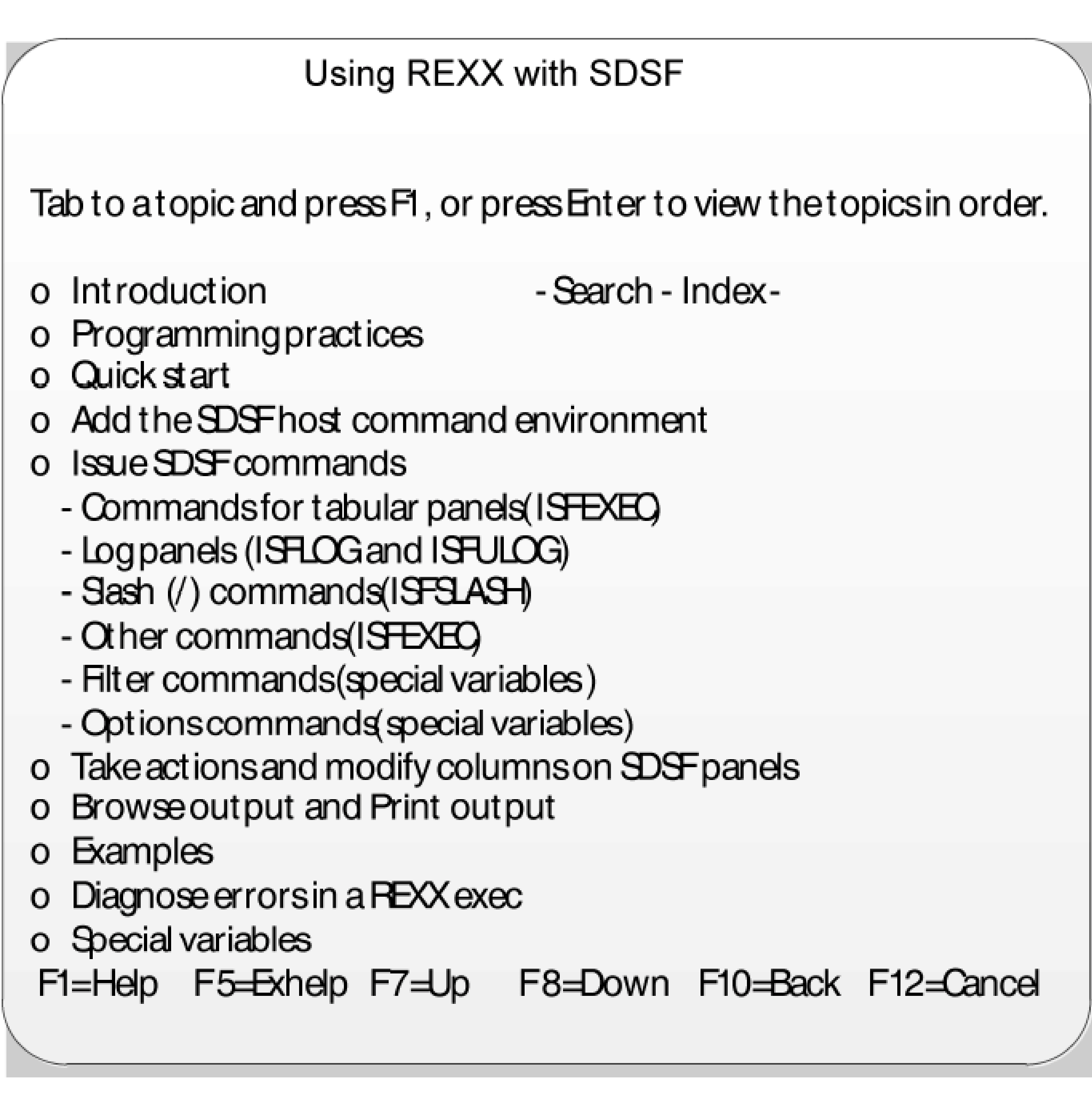


图 6-4　REXXHELP 帮助

此外，如果用户希望了解更多的 REXX 和 SDSF 交互的相关知识可以参考 IBM 白皮书 *Implementing REXX Support in SDSF*，其中详细介绍了许多 REXX 与 SDSF 交互的案例，如保存 SYSOUT 输出到 PDS 数据集、作业调度控制以及图形化显示 SDSF 数据等。SDSF 已经支持 JES3 系统，REXX/SDSF 也在其中得到了广泛应用，如果用户使用的是 JES3 而非 JES2 系统的话，可以参考 IBM 白皮书 *Using SDSF in a JES3 Environment*。

6.6 REXX 与 FTP 的交互

REXX 可以构建 TSO FTP 命令，并使用 FTP(文件传输协议)操作有关的数据集、HFS 文件、批处理作业、SPOOL 等。可以直接在 REXX 中编写 FTP 命令，但是通常情况下使用数据栈来存放这些命令是更好的做法。

下面的示例是一个简单的 REXX 与 FTP 的交互，使用账号密码登录，随后通过 DIR 命令，列出用户编目中数据集的属性。

```
/* REXX */
QUEUE 'IBMUSER'      /* INPUT USER ID HERE */
QUEUE '****'         /* INPUT USER PSW HERE */
QUEUE 'DIR'
QUEUE 'QUIT'
X = OUTTRAP(OUTPUT.)
ADDRESS TSO "FTP 127.0.0.1"
DO I = 1 TO OUTPUT.0
   SAY OUTPUT.I
END
```

与在 REXX 中编写并提交 JCL 作业相似，只需要将待执行的 FTP 命令预先存入数据栈中，并通过 OUTTRAP 函数获取执行结果。该 REXX 脚本的执行结果如图 6-5 所示。

对于较为复杂的 REXX 交互 FTP 应用，还有另一种更为合适的方式，使用 FTP 客户端应用编程接口（FTP Client API），它允许不同语言编写的应用程序向 z/OS FTP 客户端发送 FTP 子命令，并且如果应用程序以异步的方式发送请求，则该程序可以同时与多个 FTP Client API 实例交互。除此以外，FTP Client API 还允许应用程序获取包括来自客户端的消息、FTP 服务器的应答，以及请求结果等各种信息。

z/OS 操作系统从 1.6 版本时开始提供 FTP Client API，最先支持的语言包括主机汇编语言、COBOL 以及 PL/I。经过 z/OS v 1.7 的扩展，FTP Client API 可以支持 C 语言库，包括 C 及 C++编写的应用程序都能够使用 FTP 服务。从 z/OS v1.8 版本开始，REXX 也可以支持 FTP Client API。这不但意味着，用户可以通过 REXX 来调用 FTP 命令，从而进行文件的上传下载操作，同时也大大提升了 z/OS 操作系统文件上传/下载的自动化能力。图 6-6 是 REXX 实现 FTP Client API 的结构图。

```
220-FTPSERVE IBM FTP CS V1R12 at SOW1.DAL-EBIS.IHOST.COM, 13:45:59 on 2011-07-
14.
220 Connection will close if idle for more than 5 minutes.
331 Send password please.
230 IBM100S is logged on.  Working directory is "IBMUSER.".
227 Entering Passive Mode (127,0,0,1,4,22)
125 List started OK
Volume Unit    Referred Ext Used Recfm Lrecl BlkSz Dsorg Dsname
VPWRKC 3390   2011/06/26  1    3  FB      80  8000  PO   CREXX
VPD91C 3390   2011/05/28  1    3  FB      80  4000  PO   DB2.LAB.CPY
VPWRKB 3390   2011/06/06  1    3  FB      80  4000  PO   DB2.LAB.DBRM
VPD91C 3390   2011/06/26  1    3  FB      80  4000  PO   DB2.LAB.JCL
VPWRKA 3390   2011/06/10 12   14  U     4096  4096  PO   DB2.LAB.LOAD
VPWRKB 3390   2011/05/05  7    9  FB      80  4000  PO   DB2.LAB.MID
VPWRKB 3390   2011/05/05 11   13  FB      80  4000  PO   DB2.LAB.OBJ
VPD91C 3390   **NONE**    1    3  FB      80  4000  PO   DB2.LAB.OUT
VPWRKB 3390   2011/06/19  1    3  FB      80  4000  PO   DB2.LAB.SQL
VPWRKB 3390   2011/06/08  2    4  VB    4092  4096  PS   DB2.LAB.SQLRES
VPWRKB 3390   2011/06/19  3    5  FB      80  4000  PO   DB2.LAB.SRC
VPWRKA 3390   2011/06/02  1    1  FB     218 27904  PS   DB2.LAB.TRANFILE
VPWRKC 3390   2011/07/12  1    3  FB      80  4000  PO   JCL
VPWRKA 3390   2011/07/14  3    3  FB      80  4000  PO   REXX
VPWRKC 3390   2011/07/13  1    3  FB      80  8000  PO   REXX.JCL
VPWRKA 3390   2011/07/14  2    3  FB      80  6160  PO   SOW1.ISPF.ISPPROF
VPD91C 3390   2011/07/14  1    9  VA     125   129  PS   SOW1.SPFLOG1.LIST
250 List completed successfully.
221 Quit command received. Goodbye.
***
```

图 6-5　REXX 程序执行结果

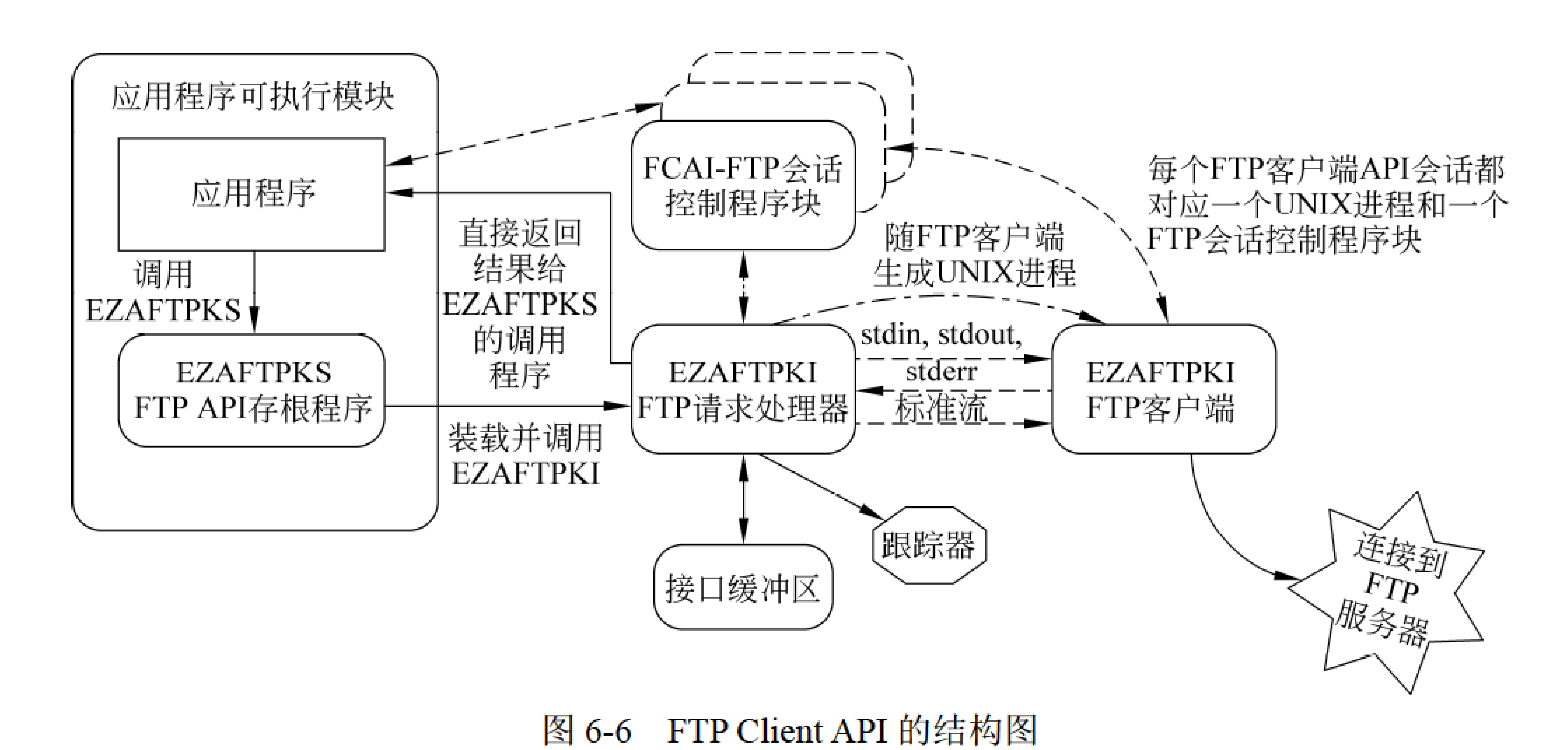


图 6-6　FTP Client API 的结构图

REXX FTP Client API 请求的格式如下。

```
rc = ftpapi(stem, request_type, parm1, parm2, ...)
```

其中，stem 参数通常是一个 REXX 的词干变量（Stem Variable，指的是复合变量的前半部分），代表了某个特定的 FTP Client 环境实例。request_type 标识了 API 的请求类型，包括如下种类；而 parm1、parm2 等参数的具体内容由特定的请求类型决定。

- create：创建新的 FTP Client API 词干变量。

- init：初始化 FTP Client API 环境。
- scmd：向 FTP Client API 发送子命令。
- poll：完成上一个 scmd 请求并返回执行结果。
- getl_find：查找并从 FTP Client 返回一行输出，这些输出通常是 init、scmd 或 poll 请求的处理结果。
- getl_copy：与 getl_find 请求类似，不同之处在于 getl_copy 返回所有的输出。
- term：终止该 FTP Client 实例。
- set_trace：允许/禁止对后续 FTP 子命令的追踪。
- set_request_timer：设置 FTP Client API 等待某个 init、scmd 或 poll 命令执行完成的时间值。
- get_fcai_map：获取 FCAI_MAP 结构的内容。

rc 变量中存放了 REXX FTP Client API 的返回值，有如下 3 种可能。

- 小于 0：表示发生错误。
- 等于 0：表示正常执行。
- 大于 0：表示正常执行，具体的返回值又包含额外信息。

图 6-7 展示了一个 FTP Client API 程序执行的主要流程，以及每个阶段发送的 FTP 请求。

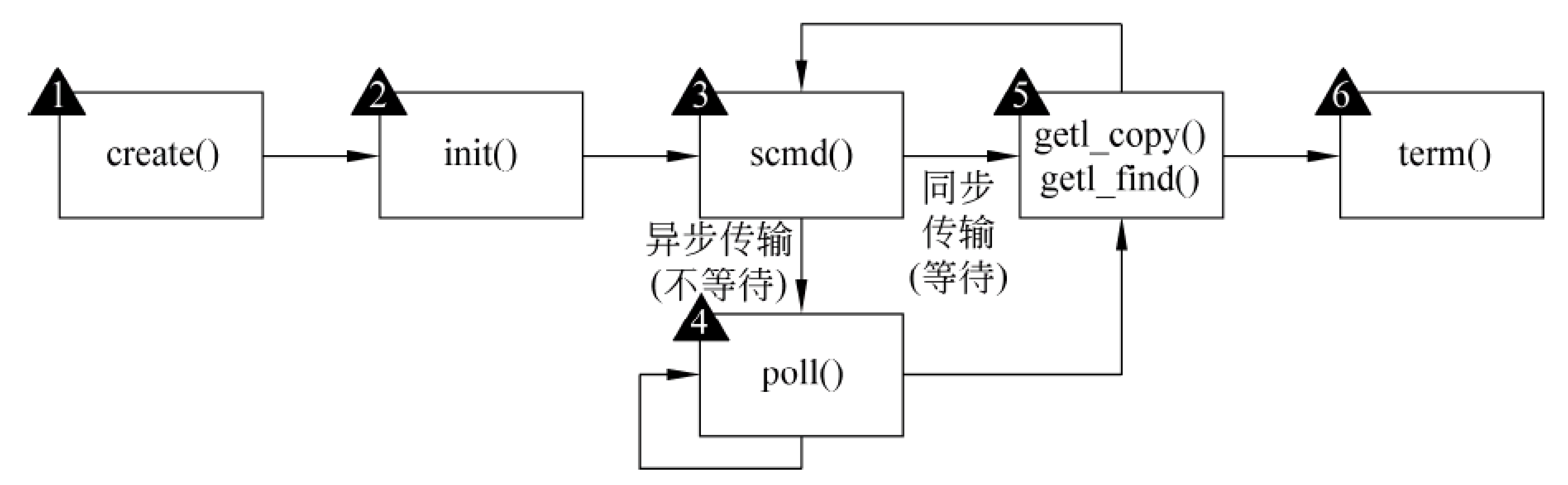


图 6-7 FTP Client API 程序执行的主要流程

同样是连接主机 FTP 服务器，并执行 DIR 命令，下面是 REXX 调用 FTP Client API 的示例。

```
/* REXX */
TRACEID = 'Z01'
INITSTR = '-w 300 127.0.0.1 21' /* CASE SENSITIVE! */
/* CREATE THE FCAI STEM. */
SAY 'INTIALIZE START'
IF FTPAPI('FCAI.', 'CREATE', TRACEID) < 0 THEN DO
  SAY 'UNABLE TO CREATE THE FCAI'
  EXIT -1
END
/* TURN ON TRACE MODE */
SAY 'TRACE ON'
RES = FTPAPI('FCAI.', 'SET_TRACE', 'ON')
```

```
/* INITIALIZE CONNETCTION */
SAY 'OPEN CONNECTION'
RES =  FTPAPI('FCAI.', 'INIT', INITSTR)
/* ENTER THE USERID. */
SAY 'INPUT USER ID'
RES = FTPAPI('FCAI.', 'SCMD', 'USER IBMUSER', 'W')
/* ENTER PASSWORD */
SAY 'INPUT PSW'
IF FCAI.FCAI_RESULT = FCAI_RESULT_PROMPTPASS THEN DO
  RES = FTPAPI('FCAI.', 'SCMD', 'PASS ****', 'W')
END
/* LIST CATALOG */
SAY 'INPUT FTP COMMAND'
RES = FTPAPI('FCAI.', 'SCMD', 'DIR', 'W') < 0
/* FETCH RESULTS */
SAY 'DIRECTORY RESULT:'
RES = FTPAPI('FCAI.', 'GETL_COPY', 'LINES.', 'L')
DO I=1 TO LINES.0
  SAY LINES.I
END
/* QUIT */
SAY 'LOGOFF'
RES = FTPAPI('FCAI.', 'SCMD', 'QUIT', 'W')
/* TERM CONNECTION */
SAY 'TERMINATE FTP CONNECTION.GOODBYE.'
RES = FTPAPI('FCAI.', 'TERM')
SAY 'GOODBYE'
EXIT
```

最需要注意的是，在初始化时参数 INITSTR，也即连接字符串是区分大小写的，如果没有注意的话，会造成错误。如果发生错误时，需要进一步了解各参数值，可以参考下面这段代码。

```
arg stem
say 'FTP Client API Error:'
say ' Result =' value(stem'FCAI_Result')
say ' Status =' value(stem'FCAI_Status')
say ' IE =' value(stem'FCAI_IE')
say ' CEC =' value(stem'FCAI_CEC')
say ' ReturnCode =' value(stem'FCAI_ReturnCode')
say ' ReasonCode =' value(stem'FCAI_ReasonCode')
```

正常情况下，该 REXX 程序的执行结果如图 6-8 所示。

如果想要了解更多关于 REXX 使用 FTP Client API 的具体信息，可以参考 IBM 白皮书 *z/OS Communications Server IP Programmer's Guide and Reference*。

```
INTIALIZE START
TRACE ON
OPEN CONNECTION
INPUT USER ID
 INPUT PSW
 INPUT FTP COMMAND
 DIRECTORY RESULT:
 Volume Unit    Referred Ext  Used Recfm Lrecl BlkSz Dsorg Dsname
 VPWRKC 3390   2011/06/26  1     3  FB       80  8000  PO  CREXX
 VPWRKC 3390   2011/10/03  1     3  FB       80  4000  PO  JCL
 VPWRKC 3390   2011/06/22  1     1  VB      255  3120  PS  LOG.MISC
 VPWRKB 3390   2011/05/12  1     3  U     32760 32760  PO  MIGLIB
 VPWRKA 3390   2011/10/03  3     3  FB       80  4000  PO  REXX
 VPWRKC 3390   2011/07/13  1     3  FB       80  8000  PO  REXX.JCL
 VPWRKB 3390   2011/06/26  1     5  VBA     125  1250  PS  REXX.PERFORM2.LIST
 VPWRKA 3390   2011/10/03  2     3  FB       80  6160  PO  SOW1.ISPF.ISPPROF
 VPD91C 3390   2011/10/03  4    36  VA      125   129  PS  SOW1.SPFLOG1.LIST
 VPWRKC 3390   2011/09/04  1     1  FB       80   800  PS  SOW1.SPFTEMP2.CNTL
 VPWRKB 3390   2011/09/04  7    59  FB      132 27984  PS  SOW1.SRCHDSL.LIST
 LOGOFF
 TERMINATE FTP CONNECTION.GOODBYE.
 GOODBYE
 ***
```

图 6-8 REXX 执行结果

6.7 REXX 与 IDCAMS 的交互

集成编目访问方法服务（IDCAMS）是 VSAM AMS 实用程序的名称，是 DFSMSdfp 提供的最重要的功能之一。IDCAMS 可以用来维护 VSAM 数据集以及管理编目，同时也可以对非 VSAM 数据集进行操作。

IDCAMS 的主要命令包括：

- DEFINE：用于定义编目、别名、数据集、次索引数据集和路径等。
- DELETE：用于删除 DEFINE 命令所建的编目、数据集、次索引数据集和路径。
- ALTER：用于更改编目中有关数据集属性的记录或原先所设定的参数。
- BLDINDEX：用于建次索引数据集。
- LISTCAT：用于显示编目中的信息。
- PRINT：用于显示 VSAM 数据集的记录。
- REPRO：用于复制数据集记录、产生编目的备用数据集、将顺序数据集转换成 VSAM 数据集；或者将 VSAM 数据集转换成顺序数据集、将记录写入 VSAM 数据集和重组 VSAM 数据集的组织。
- EXPORT：用于将 VSAM 数据集或者编目中的记录进行导出操作。
- IMPORT：用于将利用 EXPORT 命令所产生的记录导入 VSAM 数据集。

由于 IDCAMS 命令在 TSO 宿主环境下可以直接被调用，因此编写 REXX 脚本可以为 VSAM 数据集及编目的管理操作带来便利。开发者可以通过 REXX 编写 IDCAMS 命

令，并使用 OUTTRAP 函数获取命令的执行结果，甚至采用 PARSE 函数以及 EXECIO 命令，调整执行结果的格式，并将其存放到指定的数据集中。

在下面的示例中，通过 REXX 直接执行 IDCAMS 的 LISTCAT 命令，查看数据集 IBMUSER.JCL 所在的编目名称。

```
/* REXX */
X = OUTTRAP(OUTPUT.)
SAY 'START TO USE IDCAMS LISTCAT:'
"LISTCAT ENTRIES('IBMUSER.JCL')"
SAY 'COMMAND RESULT:'
DO I = 1 TO OUTPUT.0
  SAY OUTPUT.I
END
```

其中，复合变量 OUTPUT.存放了 OUTTRAP 获取的 LISTCAT 命令执行结果，如果命令执行成功，存入的即是数据集 IBMUSER.JCL 所在的编目信息；如果命令执行失败，存入的则是错误信息。

该脚本的执行结果如下，如有需要，也可以不直接显示执行结果，而是将复合变量中的内容写入到指定数据集中长期保存。

```
START TO USE IDCAMS LISTCAT:
COMMAND RESULT:
NONVSAM ------- IBMUSER.JCL
     IN-CAT --- MASTERV.CATALOG
***
```

在通常情况下，提交作业的方式来调用 IDCAMS 实用程序较为常用，在 REXX 中编写 JCL 作业，并通过 TSO/E SUBMIT 命令提交的例子在之前的示例已经出现过一次，本例中使用的也是同样的方法。

通过 REXX 以 Batch 的方式执行 IDCAMS 命令的示例代码如下。

```
/* REXX */
QUEUE "//LISTCA   JOB NOTIFY=&SYSUID "
QUEUE "//STEP1    EXEC PGM=IDCAMS "
QUEUE "//SYSPRINT DD SYSOUT=* "
QUEUE "//SYSIN  DD * "
QUEUE "  LISTCAT ENTRIES('IBMUSER.JCL') "
QUEUE "/* "
X = OUTTRAP(OUTPUT.)
ADDRESS TSO "SUBMIT * "
DO I = 1 TO OUTPUT.0
   SAY OUTPUT.I
END
```

执行该 REXX 脚本时，JES 会将作业执行的返回码显示在屏幕上。可以在 SDSF 中查看其具体信息，其部分内容如图 6-9 所示。

```
 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
   Display  Filter  View  Print  Options  Search  Help
 -------------------------------------------------------------------------------
 SDSF OUTPUT DISPLAY LISTCA   JOB06871  DSID     4 LINE 12      COLUMNS 02- 81
 COMMAND INPUT ===>                                            SCROLL ===> PAGE
        CPU:      0 HR  00 MIN  00.01 SEC    SRB:     0 HR  00 MIN  00.00 SEC
        VIRT:   476K  SYS:   260K  EXT:        OK  SYS:    11080K
IEF375I  JOB/LISTCA  /START 2011193.0814
IEF033I  JOB/LISTCA  /STOP  2011193.0814
        CPU:      0 HR  00 MIN  00.01 SEC    SRB:     0 HR  00 MIN  00.00 SEC
IDCAMS  SYSTEM SERVICES                                           TIME: 08:14:20

  LISTCAT ENTRIES('IBMUSER.JCL')
NONVSAM ------- IBMUSER.JCL
     IN-CAT --- MASTERV.CATALOG
IDCAMS  SYSTEM SERVICES                                           TIME: 08:14:20
         THE NUMBER OF ENTRIES PROCESSED WAS:
                   AIX -------------------0
                   ALIAS -----------------0
                   CLUSTER ---------------0
                   DATA ------------------0
                   GDG -------------------0
                   INDEX -----------------0
                   NONVSAM ---------------1
                   PAGESPACE -------------0
                   PATH ------------------0
                   SPACE -----------------0
                   USERCATALOG -----------0
                   TAPELIBRARY -----------0
                   TAPEVOLUME ------------0
                   TOTAL -----------------1
         THE NUMBER OF PROTECTED ENTRIES SUPPRESSED WAS 0
IDC0001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0

IDC0002I IDCAMS PROCESSING COMPLETE. MAXIMUM CONDITION CODE WAS 0
******************************** BOTTOM OF DATA ********************************




  F1=HELP       F2=SPLIT      F3=END        F4=RETURN     F5=IFIND      F6=BOOK
  F7=UP         F8=DOWN       F9=SWAP      F10=LEFT      F11=RIGHT     F12=RETRIEVE
```

图 6-9　SDSF 中查看 JCL 作业结果

另外，也可以通过 REXX 启动 AMS（Access Method Service）的交互面板，并在其中输入 IDCAMS 命令。一个简单示例的代码如下。

```
/* REXX */
ADDRESS LINK 'IDCAMS'
```

执行该 REXX 脚本即可进入 IDCAMS 系统服务面板，结果如图 6-10 所示。

使用 REXX 程序调用 AMS 面板，除了 LINK 指令以外，也可以使用 ATTACH、LOAD

或 CALL 指令。其他语言编写的程序同样可以调用 AMS 面板，如 PL/I 等。

```
LISTCAT LEVEL(IBMUSER)
 IDCAMS  SYSTEM SERVICES
TIME: 10:33:5
0         07/11/11      PAGE      1
                              LISTING FROM CATALOG --
MASTERV.CATALOG
 NONVSAM ------- IBMUSER.JCL
      IN-CAT --- MASTERV.CATALOG
 NONVSAM ------- IBMUSER.REXX
      IN-CAT --- MASTERV.CATALOG
 NONVSAM ------- IBMUSER.REXX.JCL
      IN-CAT --- MASTERV.CATALOG
……
 NONVSAM ------- IBMUSER.S0W1.ISPF.ISPPROF
      IN-CAT --- MASTERV.CATALOG
 NONVSAM ------- IBMUSER.S0W1.SPFLOG1.LIST
      IN-CAT --- MASTERV.CATALOG
 NONVSAM ------- IBMUSER.S0W1.SPFTEMP0.CNTL
      IN-CAT --- MASTERV.CATALOG
 IDCAMS  SYSTEM SERVICES
TIME: 10:33:5
0         07/11/11      PAGE      4
                              LISTING FROM CATALOG --
MASTERV.CATALOG
          THE NUMBER OF ENTRIES PROCESSED WAS:
                    AIX -------------------0
                    ALIAS -----------------0
                    CLUSTER ---------------0
                    DATA ------------------0
                    GDG -------------------0
                    INDEX -----------------0
                    NONVSAM --------------26
                    PAGESPACE -------------0
                    PATH ------------------0
                    SPACE -----------------0
                    USERCATALOG -----------0
                    TAPELIBRARY -----------0
                    TAPEVOLUME ------------0
                    TOTAL ----------------26
              THE NUMBER OF PROTECTED ENTRIES SUPPRESSED WAS 0
```

图 6-10 IDCAMS SYSTEM SERVICE 面板

关于程序调用 AMS 面板的具体信息，参看 IBM 白皮书 *DFSMS Access Method Services for Catalogs* 中 *Invoking Access Method Services from Your Program* 的部分。

6.8 REXX 与 TCP/IP 的交互

REXX 提供了丰富的 Socket API，它是一种通用灵活的底层套接字编程接口，在 TCP/IP 传输协议中使用广泛。套接字是两个应用程序通信连接的终端标识，套接字模型建立在打开/读出/写入/关闭模式上。例如，典型的面向连接的协议（如 TCP）的交互方

式通常为“三次握手”的模式，如图 6-11 所示。

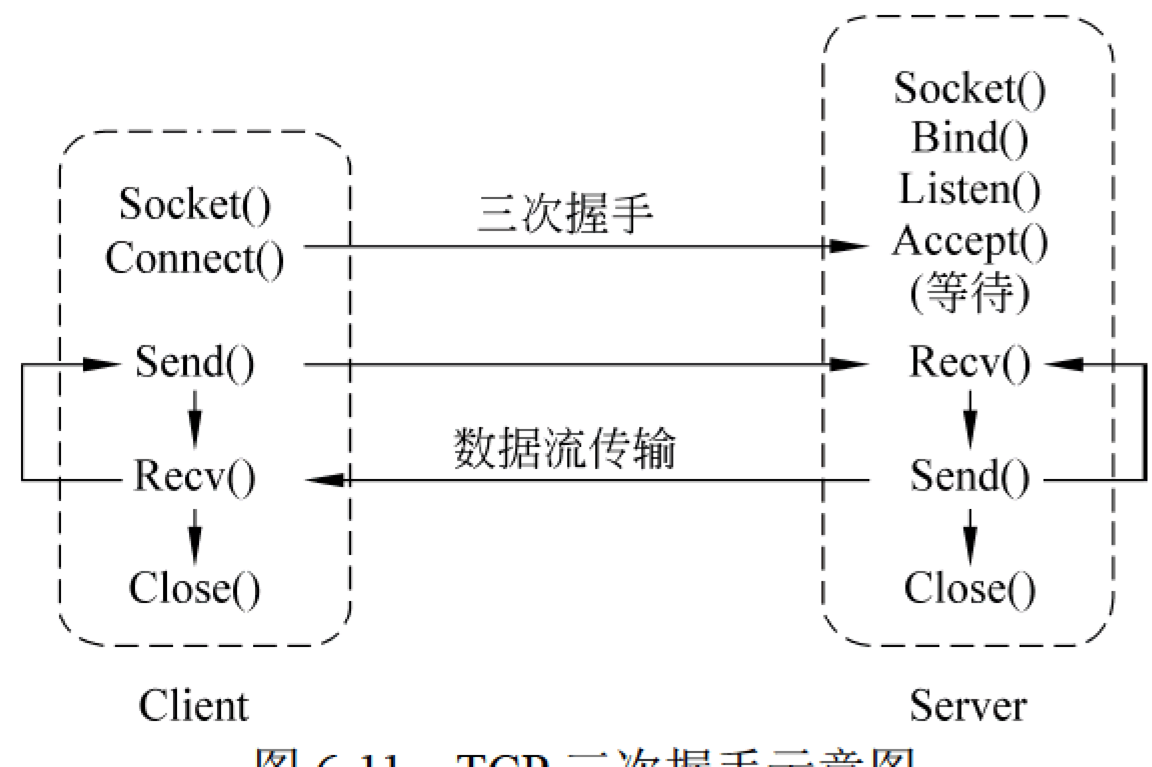


图 6-11　TCP 三次握手示意图

通过 SOCKET API，REXX 也可以和通用语言（C、Java）一样，在所提供的 TCP/IP 环境下进行有关套接字的编程。调用套接字接口的语法如下。

```
ResultString = SOCKET(Subfunction, parameters)
```

其中 Subfunction 即为所调用的套接字服务，parameters 为对应服务所需要的参数。这里需要注意的是，在这种套接字服务的调用中，REXX 的特殊变量 RC 与调用执行结果并不相关，这是其与大多数服务调用的不同之处。SOCKET API 的调用结果是以字符串的形式返回，并且该结果字符串以空格分隔开来，使返回的字符串可被 REXX 的处理函数进行解析，从而可从 ResultString 中解析出实际的返回码以及数据或者错误信息。

通常来说，返回串的首值即是调用返回码，在返回码为零的情况下，后续值即为调用具体的套接字服务所返回的信息；若返回值不为零，则返回串的第二值为发生的错误名称，后续值即为错误信息。下例为 SOCKET 服务返回信息的示例。

```
SOCKET('GetHostId')
/*调用正常返回串'0 9.4.3.2'*/
SOCKET('Recv',socket)
/*出错返回串为 '11C2 EWOULDBLOCK Problem on non-blocking socket'*/
```

通过 REXX SOCKET 可以完成的操作通常有如下几大类：

- Socket 集合处理，如 Initialize 用于定义一个 Socket 集合。
- 创建、连接、更改、关闭 Socket，例如 Socket、Bind 等。
- 交换数据，通常的数据交互操作，例如 Read/Write、Recv/Send 等。
- 解析名称或标识符，例如前面所提到的 GetHostId 返回宿主的 IP 地址等。

此外，REXX 与 TCP/IP 的交互还包括 NESTART 与 FTP 方式，NESTART 是对应的 TSO 管理命令，可通过"STACK"参数将输出放入 REXX 数据栈，并且不包含对应的头部信息（这是 OUTTRAP 所不具备的）。

6.9　REXX 与 USS 的交互

本书第 1 章中提到了 USS 脚本的使用。事实上，在 z/OS UNIX 中，同样可以使用 REXX 程序完成相应的功能。TSO/E REXX 功能的 z/OS UNIX 扩展集使得用户的 REXX 程序可以直接访问 z/OS UNIX 的可调用服务，执行相应的操作或完成对应的任务。

REXX 的 z/OS UNIX 扩展称为 syscall 命令集，其中的命令名称都对应其具体的可调服务，例如 access（判断访问文件权限）、chmod（更改文件权限）以及 chown（修改文件的所有人或所有组）等。syscall 命令集必须要得到 z/OS UNIX 系统服务的支持，否则使用 syscall 命令集的 REXX 程序是无法执行的。

此外，系统还提供了一套 z/OS UNIX REXX 功能函数集，包含了诸如标准 REXX I/O 以及一些公共文件服务和环境变量的访问方法，从而进一步扩展了 z/OS UNIX 环境下 REXX 语言的可用性。这里同样要注意的是，所有的 z/OS UNIX REXX 功能函数（除 bpxwunix()和 syscall()以外）都只能运行于 z/OS UNIX 环境之下；然而调用了 syscall 命令的 REXX 程序却可以运行于主机系统中 TSO/E、MVS、z/OS Shell 等多种环境之下（当然前提是系统安装有 z/OS USS）。

在 z/OS UNIX 系统中，带有扩展 z/OS UNIX 扩展功能集的 REXX 程序除了能够在 MVS 宿主环境下运行外，还能够被如下这几个宿主环境支持。

1. SYSCALL

SYSCALL 宿主环境支持 TSO/E 或者 MVS 批处理环境下运行的 REXX 程序调用 syscall 命令集。该宿主环境可用于支持任何调用 syscall 命令的 REXX 程序。并且无论是在 UNIX 环境还是非 UNIX 环境（TSO/MVS）下，SYSCALL 宿主环境都是可用的，此时，可以通过 syscalls()功能函数控制 SYSCALL 宿主环境，命令如下：

```
syscalls('ON')      /*建立 SYSCALL 环境*/
syscalls('OFF')     /*结束 SYSCALL 环境*/
syscalls('SIGON')   /*建立信号接口例程*/
syscalls('SIGOFF')  /*删除信号接口例程*/
```

下例中的 REXX 程序，其功能为显示当前根目录的所有信息并输出到显示终端上，程序代码都以 call syscalls 'ON'开头，用以初始化 syscall 宿主环境。

```
/* REXX */
call syscalls 'ON'
address syscall
'readdir / root. '
do i=1 to root.0
say root.i
end
```

2. SH

SH 宿主环境支持直接在 z/OS UNIX 环境下运行的 REXX 程序调用 syscall 命令集。只有当 REXX 程序是在 Shell 中运行，或者被另一个程序通过 EXEC()命令调用的情况下，REXX 程序才能使用到 SH 宿主环境。这时，由于 SH 是默认的初始宿主环境，并且在其初始化的同时也初始化了 syscall 命令集，因此此时 REXX 程序可以直接调用 syscall 命令集，而不必预先调用 syscalls('ON')命令，因为此时 syscall 命令（如 chmod）将被自动解释为 z/OS Shell 命令得到执行，其形式如下。

```
/bin/sh -c shell_command
```

在 REXX 程序中调用的 z/OS Shell 命令可以使用到所有当前打开的文件标识符以进行文件操作，同时所有环境变量对它们也是可用的。不过，由于 Shell 命令事实上是运行在 shell 进程中而非 REXX 程序的进程中，所以其无法对 REXX 环境进行操作。举例来说，在 REXX 程序中调用'cd /'这一 Shell 命令是无法改变 REXX 进程本身的路径的。

另外值得一提的是，当一个 REXX 程序从 z/OS Shell 或另一程序中开始运行，SH 和 SYSCALL 宿主环境都会被自动初始化，使其对 REXX 程序可用；而当 REXX 程序从 TSO/E 或者 MVS 批处理环境下运行时，只有 SYSCALL 宿主环境是可用的。

z/OS UNIX REXX 扩展功能函数集扩展了 z/OS 的相关功能（如标准 I/O 操作等），除去 bpxwunix()和 syscall()外，其他 z/OS UNIX REXX 只能运行于 z/OS UNIX 环境中。z/OS REXX 功能函数同样可通过子例程调用的方式实现。

下面示例中的 bpxwunix 子例程的作用和 sh –c 相似，通过将单一命令传递给 shell 进行运行。bpxwunix()功能函数的作用是运行一个 shell 命令，并提供可选的（标准输入/输出）选项。这里通过 DD 名 SYSTSPRT（由对应的 JCL 作业提供的）指定了程序处理后的输出位置。

```
/* REXX */
/* Procedure: USSCMD                      */
/* Description: Run USS shell command */
/* Format is: usscmd uss_cmd           */
Trace O
Parse Arg uss_cmd
Call Bpxwunix uss_cmd,, "DD:SYSTSPRT"
If result<>0 Then Say "Rc("!!result!!")"
Exit result
```

3. TSO

TSO 宿主环境可以在 z/OS Unix 中直接使用，因此调用了 TSO/E 命令（包括 TSO/E 命令、CLIST 以及 REXX 程序）的 REXX 程序可以在 z/OS UNIX 系统中正常运行。启用 TSO 宿主环境的 REXX 语句如下。

```
address tso [command]
```

系统将在另一个单独的地址空间中开启一个临时的 TSO 进程（称为 TSO TMP），以

支持 REXX 程序中所有 TSO 命令的执行。该 TSO 进程从 REXX 程序执行第一条 TSO/E 命令时启动，当所有 TSO/E 命令都执行完毕后，该临时进程将自动关闭，可以使用 Unix Shell 命令 ps 来查看这个进程的当前状况。如果在执行某条 TSO/E 命令时 TSO 进程异常结束，则该命令会执行失败（返回的错误码为 16），而之后的 TSO/E 命令会在新开启的 TSO 进程中继续执行。

同样必须要注意，由于 TSO/E 命令并不在 REXX 程序进程中执行，因此无法对 REXX 环境进行操作，反之亦然，在 REXX 进程中执行的其他 REXX 语句或 UNIX 命令也不能同样对 TSO TMP 进程有任何修改。

大多数的 TSO/E 命令使用 TGET()函数来读取输入的参数和内容，而输入的数据源首先是当前数据栈，随后才是 REXX 程序标准 I/O 数据流。数据栈中的所有内容都会被传递到 REXX 程序，无论其是否真正会处理这些数据，也就是说，当 TSO/E 执行完毕后数据栈会被清空。

所有 TSO/E 命令都会通过标准输出流来返回结果或进行输出，同样可以使用 OUTTRAP()函数来捕获返回的内容。REXX 保留变量 RC 依然存放程序的返回代码，如果返回值为-3，通常说明 TSO 命令没有被找到；如果返回值为 16，说明执行 TSO 命令的进程发生异常；其他的返回负值均为 Abend Code，可以通过查询相关资料来确定其对应的具体错误原因。

下面的一系列例子展示了在 z/OS UNIX 下 REXX 程序是如何执行 TSO 命令的，可以看到这与 REXX 程序在 TSO 环境内的操作并无太大区别。

```
/* 执行 TSO Time 命令 */
address tso 'time'

/* 通过 outtrap()获取执行结果并输出 */
call outtrap out.
address tso 'listc'
do i=1 to out.0
say out.i
end

/* 在 TSO/E 环境中运行一个 REXX 程序 */
address tso
"alloc fi(sysexec) da('schoen.rexx') shr"
"myexec"
```

实际上，REXX 作为系统维护不可缺少的工具，在 USS 中的使用是非常频繁的，下面的示例中，REXX 程序读取并打印系统的根目录。

```
/* rexx */
call syscalls 'ON'
address syscall
'readdir / root.'
```

```
do i=1 to root.0
   say root.i
end
```

下面的示例中，REXX 程序打开并读取了一个文件。

```
/* rexx */
call syscalls 'ON'
address syscall
path='/u/schoen/my.file'
'open (path)',
   O_rdonly,
   000
if retval=-1 then
do
   say 'file not opened, error codes' errno errnojr
   return
end
fd=retval
'read' fd 'bytes 80'
```

下面的示例中，REXX 程序打开一个文件并向其中写入内容。

```
/* rexx */
call syscalls 'ON'
address syscall
path='/u/schoen/my.file'
'open' path,
   O_rdwr+O_creat+O_trunc,
   660
if retval=-1 then
do
   say 'file not opened, error codes' errno errnojr
   return
end
fd=retval
rec='hello world' || esc_n
'write' fd 'rec' length(rec)
if retval=-1 then
   say 'record not written, error codes' errno errnojr
'close' fd
```

更多有关 REXX 与 z/OS UNIX System Services 交互的内容，包括 syscall 命令集和 z/OS UNIX REXX 函数的具体语法描述，以及其他在 USS 中使用 REXX 的实例，可以参阅 IBM 白皮书 *Using REXX and z/OS UNIX System Services*。

6.10　REXX 与 CICS 的交互

早期，CICS 对于 REXX 的支持由两个单独的产品实现，分别是 REXX Development System for CICS/ESA 和 REXX Runtime Facility for CICS/ESA。现在，这两个产品都已经被整合到 CICS Transaction Server 中。

在 CICS TS 上运行的 REXX 程序可以使用以下宿主环境。

- REXXCICS：默认的 REXX/CICS 宿主环境。
- CICS：执行 CICS 命令（如 SEND 命令、RECEIVE 命令）时使用的宿主环境。
- EXECDB2：执行 DB2 命令（如 DISPLAY 命令）时使用的宿主环境。
- EXECSQL：通过 CICS/DB2 接口执行 SQL 语句（如 SELECT）时使用的宿主环境。
- EDITSVR：创建 EDIT 会话使用的宿主环境。
- FLSTSVR：执行文件列表实用程序（File List Utility）相关命令时，使用的宿主环境。
- RFS：执行 REXX 文件系统（REXX File System）命令时使用的宿主环境。
- RLS：执行 REXX 列表系统（REXX List System）命令时使用的宿主环境。

REXX/CICS 为 REXX Development System 提供的主接口程序为 CICREXD，为 REXX Runtime Facility（主要用于在 CICS Region 中加载调用 REXX 程序）提供的主接口程序为 CICREXR。每一个 REXX 程序都在单独的 CICS Task 下运行，而嵌套的 REXX 程序则在其父程序所在的 CICS Task 下运行。使用 GETVERS 命令可以判断某个 REXX 程序的运行环境是 REXX Development System 还是 REXX Runtime Facility。

1. 启动 REXX 程序

在 CICS 上启动一个 REXX 程序有许多种方法，最常用的是通过终端直接输入，就像平时输入 Transaction ID 直接运行一个与之关联的程序一样。

CICS 通过 Transaction ID 绑定程序来决定执行哪一个程序，而 REXX/CICS 通过由 DEFTRNID 命令创建的一张表，将 Transaction ID 和 REXX 程序关联到一起。REXX/CICS 默认的 Transaction ID 是 REXX，而其对应的默认 REXX 程序为 CICRXTRY。即如果在 CICS 屏幕中输入 REXX，CICRXTRY 程序则被启动。如果在 Transaction ID REXX 之后还跟随了其他参数，如“REXX MYEXEC AAA”，则名为 MYEXEC 的 REXX 程序会被启动，而 AAA 则会作为参数传入该程序中。

另一种调用 REXX 程序的方式是通过使用 CICS START 命令。通过 START 命令确定 Transaction ID，随后通过 REXX/CICS 关联表确定被调用的 REXX 程序。当这个被调用程序指定为 CICRXTRY 时，跟随在 CICS START 命令后的第一个参数将会作为实际被启动的 REXX 程序，如“START MYEXEC AAA”，实际上启动的是 MYEXEC 程序，而 AAA 作为参数传入。如果 START 命令后没有跟任何参数，则依然启动 CICRXTRY 程序。当该被调用程序被指定为 CICRXTRY 以外的程序时，则所有 CICS START 命令后的内容

都会作为参数传入被调用的 REXX 程序。

其他启动 REXX 程序的方法还有：使用 CICS LINK 或 XCTL 命令；通过 OfficeVision/MVS 调用 REXX 程序等。读者如果有兴趣，可以参阅 IBM 白皮书 *REXX for CICS/ESA R1V1 Guide and Reference*，以了解更多内容。

2. 装载 REXX 程序

当系统查找并装载一个 REXX 程序时，会按以下步骤操作：

（1）在内存中搜索是否有已经通过 EXECLOAD 命名装载完毕的该 REXX 程序，如果有的话，则直接使用该程序的副本。

（2）如果在内存中没有找到，则系统会查找用户当前的 RFS 目录下是否有相关 REXX 程序。RFS 的全称是 REXX File System，是基于 VSAM 的文件系统，可以由 REXX/CICS 提供，也可以由 MVS 的分区数据集支持。RFS 目录可以通过 REXX/CICS 的 CD 命令来定义。

（3）如果依然没有找到，则系统会查找用户通过 REXX/CICS PATH 命令定义的 Path 路径。

（4）如果用户是授权用户（authorized user），则系统会去查找 CICS 的启动 JCL 文件中 CICAUTH 参数指定的数据集，如果在该数据集中找到对应的 REXX 程序，则将其装载入内存。

（5）如果通过之前的路由仍未找到对应的程序，则系统会依次查找 CICEXEC 和 CICUSER 参数对应的数据集，如果依然没有找到，则返回错误码。

3. 编辑 REXX 程序

如果安装了 REXX Development System 的话，则用户可以通过 REXX/CICS 提供的编辑器在 CICS 中编辑 REXX 程序。当然如果 REXX 程序存放在 MVS 的分区数据集中的话，也可以通过 ISPF/PDF 编辑器在 TSO 环境下进行编写或修改，就如本书的绝大部分章节中介绍的一样。

4. 定义用户自定义命令

在 TSO 环境下，将 REXX 程序放在系统库（SYSPROC 或 SYSEXEC）中的话，可以通过在命令栏中直接输入该脚本名称的方式来隐式调用 REXX 程序，就像输入 TSO 命令一样（详见第 8 章）。

类似地，在 CICS 中，如果用户想要如同执行 REXX/CICS 命令一样直接调用某个 REXX 程序的话，可以通过 DEFCMD 命令来定义自己的 REXX/CICS 自定义命令。

5. 伪会话交易（Pseudo-Conversational）支持

伪会话交易是 CICS 的重要特性之一，允许客户端与服务器之间以异步的方式交互，从而节省了不必要的等待时间，提高了系统性能。图 6-12 是关于会话（conversational）交易和伪会话（pseudo-conversational）交易的过程比较。

CICS 的伪会话特性也能够支持 REXX，通过使用 CICS 的 RETURN TRANSID()命令就可以返回对应的 Transaction ID。如果要使用伪对会话特性，首先需要通过

REXX/CICS 的 PSEUDO 命令将伪会话模式开启：

```
PSEUDO ON
```

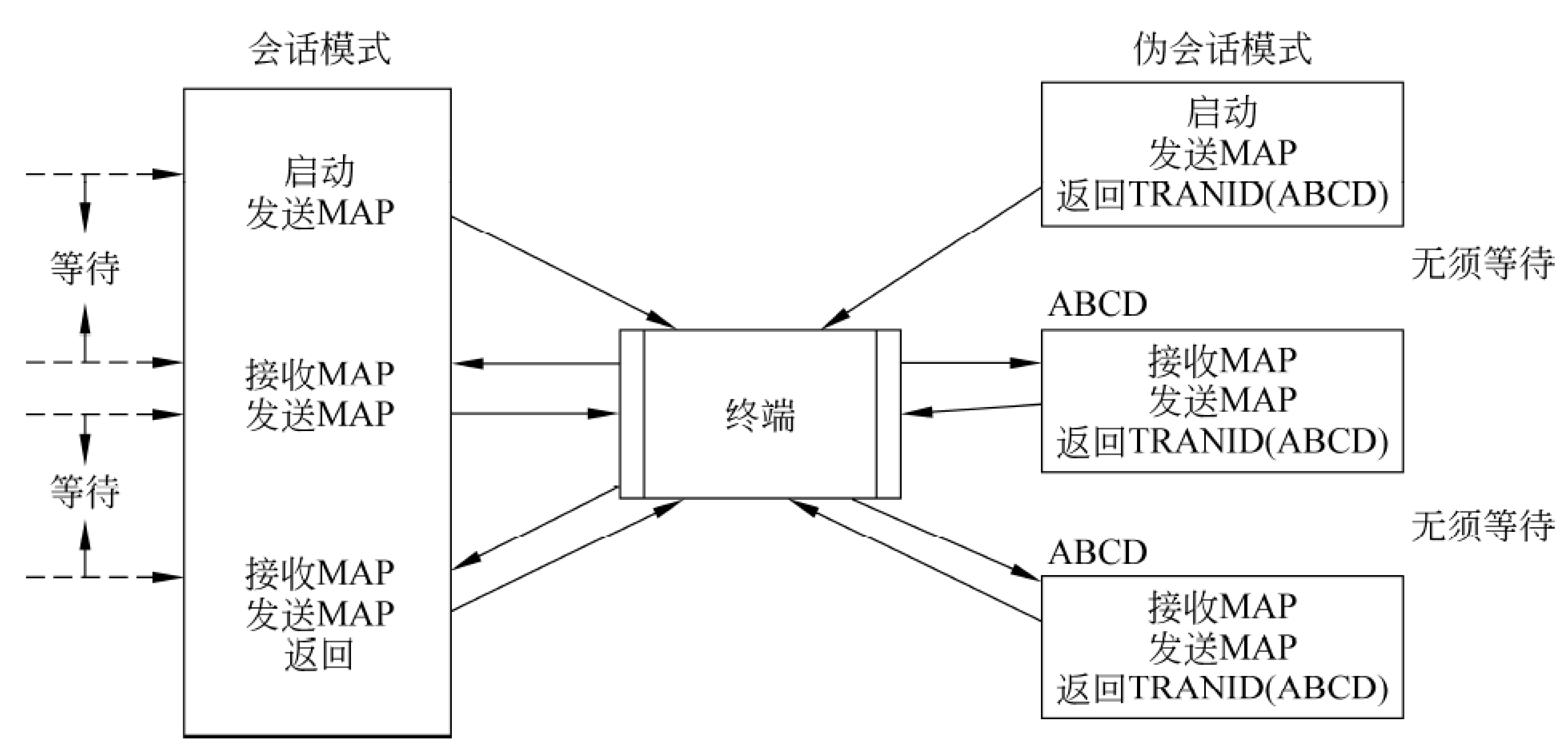


图 6-12　会话交易和伪对话交易对比

该命令一旦执行，REXX 程序将立即切换到伪会话模式，CICS 会执行所有的文件及数据库修改，随即清空所有下一伪会话终端会读取的非共享 GETMAIN 域。伪会话模式的开启和关闭是暂时的，只在一个 REXX 程序（包括其嵌套子程序）的执行范围内起作用。当前 REXX 程序结束运行后，伪会话模式又会恢复到其初始设置的状态。如果要改变伪会话模式的初始状态，可以使用 SETSYS PSEUDO 命令。

6. 调用 DB2 接口

REXX/CICS DB2 接口提供了使用 REXX 程序执行 SQL 语句以及 DB2 命令的一系列支持。其中，SQL 语句是动态执行的。而 DB2 命令则是通过使用 DB2 IFI（Instrumentation Facility Interface）来调用的。通过预先定义好的 REXX 变量，REXX/CICS DB2 接口将 SQL 语句和 DB2 命令执行的结果返回给用户。

如果要将 SQL 语句或 DB2 命令嵌入 REXX 程序，需要切换到相应的宿主环境。对于 SQL 语句，需要预先切换到 EXECSQL 宿主环境；而对于 DB2 命令，则需要预先切换到 EXECDB2 宿主环境。

（1）执行 SQL 语句

在 EXECSQL 宿主环境下，不是所有的 SQL 语句都可以直接执行。下列 SQL 语句就不能在此环境下使用。

```
BEGIN DECLARE SECTION / CLOSE / COMMIT / CONNECT / DECLARE CUSOR / DECLARE
STATEMENT / DECLARE TABLE / DESCRIBE / END DECLARE SECTION / EXECUTE /
EXECUTE IMMEDIATE / FETCH / INCLUDE / OPEN / PREPARE / ROLLBACK / SET CURRENT
PACKAGESET / SET HOST VARIABLE / WHENEVER
```

而下列 SQL 语句在 EXECSQL 环境下可以直接被执行。

```
ALTER / CREATE / COMMENT ON / DELETE / DROP / EXPLAIN / GRANT / INSERT /
LABEL ON / LOCK / REVOKE / SELECT / SET CURRENT SQLID / UPDATE
```

另外，在 EXECSQL 环境下执行 SQL 语句不能使用宿主变量，但是用户可以使用 REXX 变量来代替宿主变量进行参数传递。

SQL 语句的执行结果会被返回到预定义的 REXX 变量中。如返回码会被存入变量 RC，而关于 SQL 语句具体执行的情况，会被存放到变量 SQLCA 中，也即 SQL COMMUNICATION AREA 中。SELECT 语句返回的数据集会被存入对应变量 column.n 中，column.n 是复合变量，其名称（column）对应了数据表的列名，n 则对应了返回的特定行。

（2）执行 DB2 命令

自 DB2 v2.3 版本开始，下列 DB2 命令都可以在 EXECDB2 环境下执行。

```
-ARCHIVE LOG
-CANCEL DDF THREAD
-DISPLAY DATABASE
-DISPLAY LOCATION
-DISPLAY RLIMIT
-DISPLAY THREAD
-DISPLAY TRACE
-DISPLAY UTILITY
-MODIFY TRACE
-RECOVER BSDS
-RECOVER INDOUBT
-START DATABASE
-START DDF
-START RLIMIT
-START TRACE
-STOP DATABASE
-STOP DB2
-STOP DDF
-STOP RLIMIT
-STOP TRACE
-TERM UTILITY
```

而自 DB2 v3.1 版本后，下面这些 DB2 命令也被 EXECDB2 环境所支持。

```
-ALTER BUFFERPOOL
-CANCEL DDF THREAD
-DISPLAY ARCHIVE
-DISPLAY BUFFERPOOL
-RESET INDOUBT
-SET ARCHIVE
```

然而，仍有些 DB2 命令不能被使用，如-START DB2（该命令只能在 MVS 控制台中执行）。另外，虽然 STOP DB2 可以被使用，但建议用户不要使用这条命令，以免造成不必要的麻烦。

如果要为 DB2 命令传递参数的话，同样可以使用 REXX 变量。而 DB2 命令执行的

返回结果同样也会存放在预定义的 REXX 变量中，如 RC 等。

关于 SQL 语句以及 DB2 命令在 REXX/CICS 环境下更多的使用指导，请参考 IBM 白皮书 *REXX for CICS/ESA R1V1 Guide and Reference*。

6.11　REXX 与 DB2 的交互

IBM DB2 REXX Language Support 提供了使用 REXX 语言编写 SQL 应用程序的支持，REXX 程序同样可以像用户通常编写的 COBOL 程序一样访问 DB2 进行数据库操作，并且 DB2 也支持用 REXX 语言编写的存储过程。

与其他语言不同的是，所有 REXX 中执行的 SQL 语句都是动态 SQL 语句。所以，当系统安装 DB2 REXX Language Support 后，相当于绑定了 4 个连接 DB2 的 Package，每个 Package 均有各自不同的隔离级别（如表 6-1 所示，由高到低排列）。

表 6-1　DB2 的 Package 和隔离级别

Package 名称	隔离级别	Package 名称	隔离级别
DSNREXRR	**RR**（Repeatable Read）	DSNREXCS	**CS**（Cursor stability）
DSNREXRS	**RS**（Read stability）	DSNREXUR	**UR**（Uncommitted read）

如果想要更改 REXX SQL 程序的隔离级别，可以执行 SET CURRENT PACKAGESET 语句，更改对应的 Package 来进行切换。下例中便将隔离级别切换为 RR。

```
"EXECSQL SET CURRENT PACKAGESET='SNREXRR'"
```

DB2 REXX Language Support 包含了如下应用编程接口。

- CONNECT：将 REXX 程序连接到一个 DB2 子系统。
- DISCONNECT：将 REXX 程序与某个 DB2 子系统的连接断开。
- EXECSQL：执行 SQL 语句。所有即将执行的 SQL 语句都应该紧跟在 EXECSQL 之后出现；或者存放在 REXX 变量中，而该变量同样必须紧跟在 EXECSQL 之后。下面这两种写法都是正确的。

```
EXECSQL "COMMIT"

rexxvar="COMMIT"
EXECSQL rexxvar
```

这些编程接口都必须在特定的宿主环境 DSNREXX 下才能使用，因此如果要执行访问 DB2 的 REXX 程序，首先要确保 DSNREXX 环境在 REXX 的宿主环境表中，可以使用 REXX 函数 RXSUBCOM 将 DSNREXX 添加到宿主环境表中。

1. 将 SQL 语句嵌入 REXX 程序

DB2 REXX Language Support 允许大部分 DB2 for z/OS 支持的 SQL 语句执行，但下列语句不被支持。

- BEGIN DECLARE SECTION
- DECLARE STATEMENT
- END DECLARE SECTION
- INCLUDE
- SELECT INTO
- WHENEVER.

由于 REXX 语言仅支持动态 SQL，因此直接带有宿主变量的 SQL 语句如 SELECT、INSERT、DELETE 或 UPDATE 都是行不通的。正确的做法是首先执行 PREPARE 语句，在其中用参数标记替代宿主变量；随后再执行 EXECUTE 或 OPEN 或 FETCH 语句。下面这个例子中，标记?在 PREPARE 语句中代替了宿主变量 MY，当 MY 被赋值以后，使用 USING 参数将其加入 EXECUTE 语句中对应的位置。

```
SQLSTAT = "UPDATE MYTAB SET MYCOL = ? WHERE URCOL = 'YES'"
EXECSQL "PREPARE S100 FROM :SQLSTAT"
MY='YES'
EXECSQL "EXECUTE S100 USING :MY"
```

2. 宿主变量及数据类型

宿主变量在 REXX 语言中不必事先声明，这是 REXX 语言本身的特性。在 SQL 语句中使用 REXX 变量作为宿主变量时，同其他语言一样，要在变量名前加上冒号，如 ":myvar"。简单变量或者复合变量都可以作为宿主变量来使用，DB2 Language Support 会在 DB2 执行 SQL 语句之前，先行识别宿主变量。下例中，传递给 DB2 的复合变量实际上是":myvar.1.2"。

```
var1=1
var2=2
EXECSQL "OPEN C1 USING :myvar.var1.var2"
```

REXX 语言中只有一种数据类型（字符串）。因此当 REXX 程序将输入数据关联到 DB2 中数据表的某一列上时，DB2 将自动将字符串类型转化为该列本身的数据类型。而当 REXX 程序从数据表中取出数据时，DB2 同样也会将原本的数据类型转化为字符串类型。

除了让 DB2 自行判断数据类型以外，也可以通过使用 SQLDA 来告知 DB2 输入数据对应的具体类型。关于转换数据类型的具体信息，请参考 IBM 白皮书 *DB2 for z/OS Application Programming and SQL Guide*。

如果没有预先给宿主变量赋值，就将其和数据表某一列相关联，则 DB2 会返回报错信息。

当从 DB2 返回的某列为空值时，DB2 会将其关联的宿主变量所对应的指示变量（indicator variable）值置为负值，这样就说明该宿主变量为空。同样，当用户将空值传递给 DB2 时，也必须将其关联宿主变量对应的指示变量置为负数。REXX 语言与其他语言的不同之处在于，在用户给 DB2 传递空值时，不但要将指示变量置为负值，同时也要给

宿主变量本身赋值。换言之，宿主变量不能为空，例如：

```
SQLSTAT = "UPDATE MYTAB SET URCOL = ? ? "
UR = "000"
URI = -1
"EXECSQL PREPARE S1 FROM :SQLSTAT"
"EXECSQL EXECUTE S1 USING :UR :URI"
```

3. 处理结果集

REXX 程序中同时也不支持 SELECT INTO 语句，如果要取出 SELECT 语句的查询结果，必须要预先 PREPARE 查询声明语句，并为该语句打开一个游标，随后将返回的结果行通过宿主变量或 SQLDA 取出。下面的示例便说明了如何通过 SQLDA 取出 DB2 数据表的查询结果。

```
SQLSTAT = 'SELECT ID, MYCOL, URCOL FROM MYTAB ',
EXECSQL DECLARE C1 CURSOR FOR S1
EXECSQL PREPARE S1 INTO :OUTSQLDA FROM :SQLSTAT
EXECSQL OPEN C1
Do Until(SQLCODE <> 0)
  EXECSQL FETCH C1 USING DESCRIPTOR :OUTSQLDA
  If SQLCODE = 0 Then Do
    Line = ''
    Do I = 1 To OUTSQLDA.SQLD
      Line = Line OUTSQLDA.I.SQLDATA
    End I
    Say Line
  End
End
```

4. 游标和声明

在 REXX SQL 程序中，只能够使用预定义的一系列 REXX 变量来存放游标和声明（prepared statement）。

（1）C1–C100：可以用作 DECLARE、OPEN、CLOSE、FETCH 语句中的游标。其中 C1～C50 是为没有定义 WITH HOLD 选项的游标所准备的；而 C51～C100 则是为定义了 WITH HOLD 选项的游标所准备的。

（2）C101–C200：调用存储过程取回返回的结果集时，可以用作 ALLOCATE、DESCRIBE、FETCH、CLOSE 语句中的游标。

（3）S1–S100：用于 DECLARE STATEMENT、PREPARE、DESCRIBE、EXECUTE 语句的声明。

不要使用除了这些预定义 REXX 变量以外的宿主变量名称作为游标和声明的名称。在 DECLARE CUSOR 语句中把游标名称和声明名称关联起来的时候，两者的编号必须一致，例如：

```
EXECSQL 'DECLARE C1 CURSOR FOR S1'
```

5. 检测错误和警告

由于 REXX 不支持 WHENEVER 语句，通常使用下面两种方法来检测 REXX 程序与 DB2 交互时发生的错误和警告信息。

第一种方法是在每次 EXECSQL 调用后检查 SQLCODE 或 SQLSTATE 以及 SQLWARN 的值，但是这种方法无法检测到从 REXX DB2 接口连接到 DB2 之间发生的错误。

第二种方法是在每次 EXECSQL 调用后检查 REXX 的 RC 变量返回值值，如果返回值为 0 则说明没有发生 SQL 错误或警告；如果返回值为 1 则说明有警告出现；如果返回值为-1 则说明发生了 SQL 错误。同时，也可以使用 REXX SIGNAL ON ERROR 以及 SIGNAL ON FAILURE 关键字指令来检测 RC 返回值并将其转化为错误信息。

6.12 REXX 与其他编程语言

REXX 可以与其他编程语言编写的程序互相交互，如主机汇编语言（assembler）、COBOL、PL/1、C 以及 FORTRAN 等在主机平台上经常使用的语言。可以选择的交互方式有以下几个。

（1）外部函数及子例程：除了用户可以编写外部函数和子例程来扩展 REXX 的使用性能。外部函数和子例程都可以用 REXX 编写，甚至可以将其编译，然后调用 Assembler、COBOL、PL/1、C 或 FORTRAN 程序（前提条件是运行在 31 位的地址空间中）。

（2）函数包（function packages）：函数包将多个外部函数或子例程整合到一起。REXX 语言解释器在执行程序时，会首先搜索函数包，随后再搜索装载程序库（load libraries）以及执行程序库（EXEC Libraries，如 SYSEXEC 或 SYSPROC）。因此，如果将经常执行的外部函数或子例程放在程序包内，可以提高语言解释器的检索效率，从而提升其执行性能。

（3）变量获取（variable access）：REXX 提供了让其他语言编写的程序直接访问并操作 REXX 程序中变量的接口。用户可以调用 IRXEXCOM 或 IKJCT441 例程，从而对 REXX 变量进行复制、赋值、删除、获取变量名、设置变量名以及获取下一个变量等多种操作。

（4）特定宿主环境：通过 IRXSUBCM 子例程可以修改宿主命令环境表中的信息，用户动态添加自定义的宿主命令环境。

（5）其他程序调用 REXX 程序：IRXEXEC 子例程使得其他语言编写的程序能够调用 REXX 程序执行器，传递参数随后执行某个 REXX 程序，并且获得返回结果。

（6）访问数据栈：其他语言编写的程序也可以访问 REXX 数据栈，因此，数据栈可以作为 REXX 与非 REXX 程序间传递数据的手段被使用。IRXSTK 是系统提供的数据栈子例程，用于处理各种数据栈请求。

关于以上这些交互方式的更多信息，可以参阅 IBM 白皮书 *TSO/E REXX Reference*，查看相关子例程的具体参数及用法。

6.13　REXX 与其他 IBM 产品

事实上，很多 IBM 相关产品都支持 REXX，除了已经提到的 RACF、IDCAMS，还有 IPCS 等产品。由于这些产品集成于 z/OS，因此使用 REXX 构建其命令能得到良好的应用。除此之外，中间件产品 CICS、数据库产品 IMS、DB2 等，以及 z/OS 前身 VSE、OS 390 等能都良好地支持 REXX 运行（部分产品已在前面有所介绍）。REXX 本身良好的脚本语言特性，使其在各种平台的产品中都有广泛的应用。

第 7 章 REXX 与 ISPF 交互

7.1 ISPF 和 ISPF 会话

7.1.1 什么是 ISPF

ISPF（Interactive System Productivity Facility）运行在 TSO 上，是 TSO 的扩展系统，它是一种采用菜单驱动模式的用户交互工具，提供人机交互的处理。ISPF 提供如下服务。

（1）显示服务。

（2）文件定制服务。

（3）变量服务。

（4）表服务。

（5）综合性服务。

（6）会话测试组件服务，包括：

- 设置断点；
- 追踪会话服务和 Dialog 变量的使用；
- 浏览 ISPF 日志数据集的 TRACE 输出；
- 检查更新 ISPF 表；
- 交互地调用大多数的会话服务。

ISPF 提供了基于图形面板的用户界面，用户可以通过该界面上面板的目录选择，使用对应的 ISPF 的各种功能。某系统 ISPF 提供的用户主界面如图 7-1 所示。

此界面有三个主要区域。

（1）Action bar：屏幕上方的区域。选择任意一个 Action bar 都会显示一个下拉窗口。涵盖了 ISPF 的所有功能和设置。

（2）功能选项：屏幕的主体区域。这个区域显示了 ISPF 的主要功能选项菜单。直观地提供选择，方便用户使用。

（3）动态状态区域：屏幕右侧的区域。用户可以指定此区域显示的内容。可以通过操作栏中的状态栏进行修改。

ISPF 界面下方一般会标识功能键：F1 键查看帮助，F2 键分屏 F3 键退出 F8 键来向下翻页，F7 键向上翻页，F10 键向左移动，F11 键向右移动。

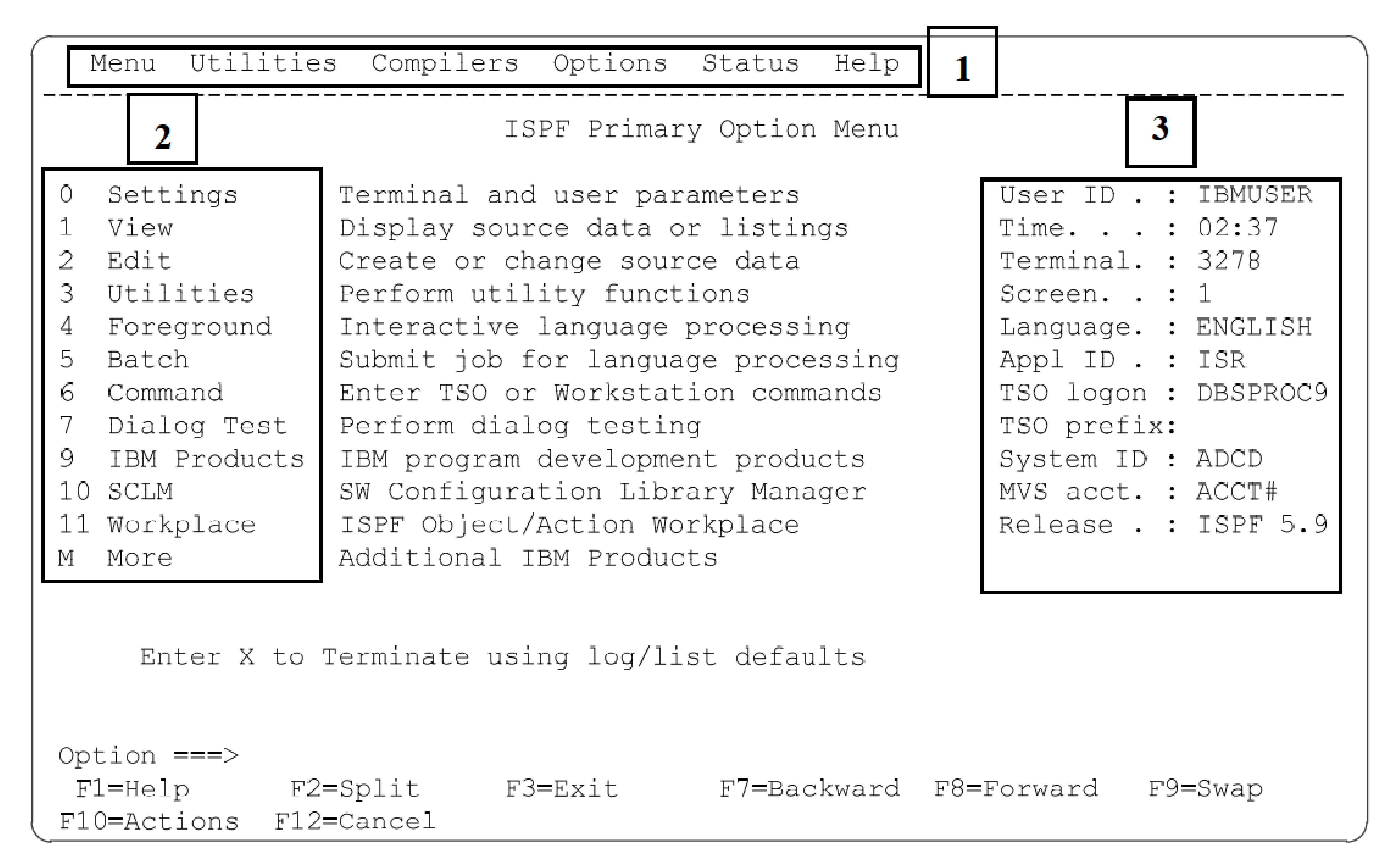


图 7-1　ISPF 主界面

在 Action bar 中，选择任意一个 Action bar 都会显示一个下拉窗口，如图 7-2 所示。

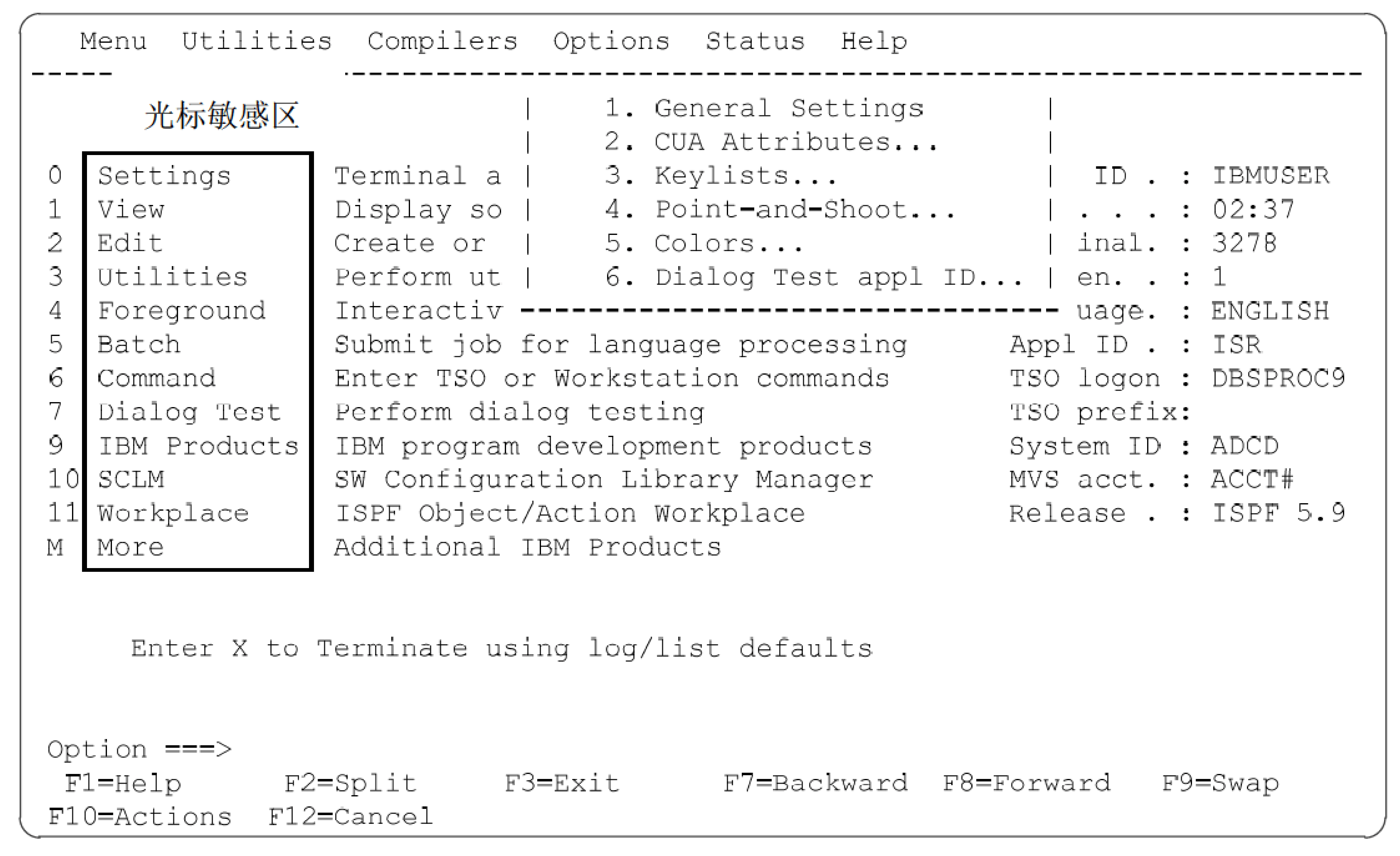


图 7-2　Action Bar 下拉窗口

功能选项的方框区域为光标敏感区（实际显示为蓝色的区域），将光标移到蓝色区域，单击，即可查看提供的子菜单选项或进入某个功能。

7.1.2 ISPF 会话

一般而言，会话（session）可理解为通过特定显示终端与计算机程序的交互，ISPF 会话特指运行于 ISPF 下的交互式对话（dialog）程序。

ISPF 会话包含相应的会话要素，主要包括功能指示、面板定义、消息定义、表格、文件裁剪框架、Dialog 变量等要素。“功能指示”决定了对话框的处理顺序，它可以调用 ISPF 会话的有关服务显示面板或消息，构建或保存表格，产生输出数据集以及控制操作模式等。通常“功能指示”可由 REXX 等脚本语言编写，也可以其他诸如 PL/I、COBOL、FORTRAN 等编程语言编写完成。下面主要介绍 REXX 与 ISPF 会话的交互。

7.1.3 ISPF 对话框定义

想要理解 ISPF 会话，首先要理解什么是 ISPF 对话框。一个对话框是人机之前的交互媒介。能够在人与计算机之间传递信息。图 7-3 就展示了 ISPF 的一个对话框。

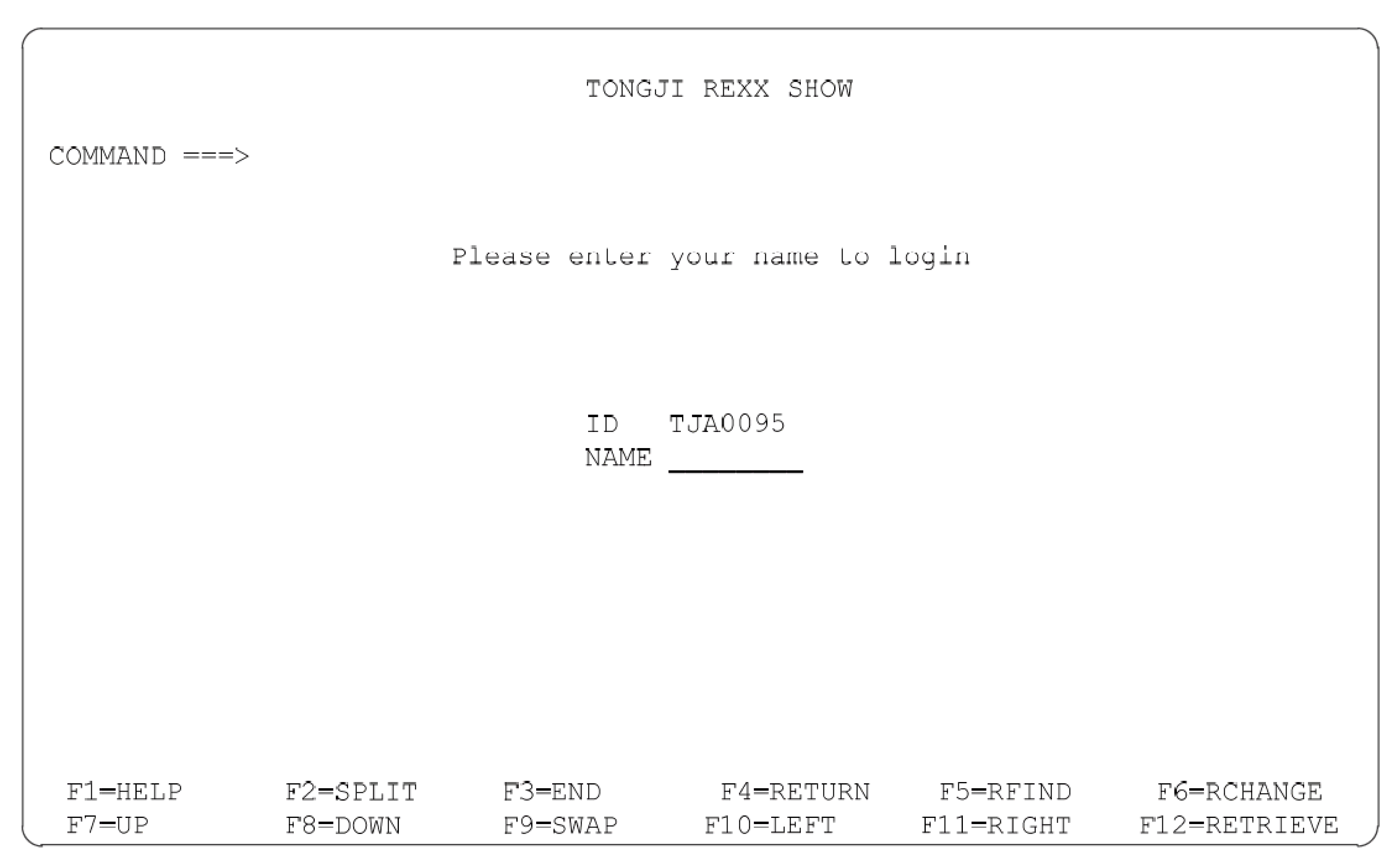


图 7-3 ISPF 对话框

图 7-3 所示的对话框对应的窗口定义代码如图 7-4 所示。

7.1.4 ISPF 对话元素

组成一个 ISPF 对话程序的元素有：功能函数、变量、命令表、面板定义、消息定义、文件裁剪框架、数据库的表。一个对话应用不必包含所有种类的元素。下面介绍几个主要对话元素。

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
 -------------------------------------------------------------------------------
 VIEW       TJA0095.REXX.PANEL(TP01) - 01.00                 Columns 00001 00072
 ****** **************************** Top of Data ****************************
 000001 )ATTR
 000002   % TYPE(TEXT) COLOR(WHITE) CAPS(OFF) SKIP(ON)
 000003   * TYPE(TEXT) CAPS(OFF) INTENS(HIGH) COLOR(GREEN) SKIP(ON)
 000004   ! TYPE(TEXT) CAPS(OFF) INTENS(HIGH) COLOR(BLUE) SKIP(ON)
 000005   _ TYPE(INPUT) PAD('_') INTENS(HIGH) COLOR(TURQ)
 000006   @ TYPE(NEF) CAPS(ON) PADC(USER)
 000007   # TYPE(OUTPUT) PAD(' ') CAPS(OFF) COLOR(YELLOW) JUST(LEFT)
 000008   ? TYPE(NT)
 000009 )BODY WIDTH(80) CMD(ZCMD) EXPAND(//)
 000010 *
 000011 !/ /TONGJI REXX SHOW/ /
 000012 *
 000013 *COMMAND ===>@ZCMD
 000014 *
 000015 *
 000016 %/ /Please enter your name to login/ /
 000017 *
 000017 *
 000018 *
 000019 *
 000020 *
 000021 */ /ID  #uid     * / /
 000022 */ /NAME_uname   * / /
 000023 *
 000024 )PROC
 000025  VER(&uname,NB)
 000026 )end
****** ************************** Bottom of Data ****************************
 Command ===>                                                  Scroll ===> PAGE
  F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
  F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

图 7-4　对话框定义

1. 功能函数

功能函数是一个命令过程（command procedures）或程序，执行处理用户请求。它可以显示面板和消息，调用 ISPF 对话框服务，建立和维护表，生成输出数据集，并控制操作模式等。功能函数可以用命令过程语言编写，如 CLIST 或 REXX，也可以使用其他语言，如 PL/I、COBOL、FORTRAN、APL2、Pascal 或者 C。

2. 变量

ISPF 服务使用变量在对话应用的各个元素之前传递信息。ISPF 提供了管理变量的一组服务。变量存储信息最大长度为 32KB，这些信息依据其使用情况被存储在变量缓冲池中。字母 Z 开头的一组变量为系统变量。以 Z 开头的变量都被保留下来供 ISPF 系统来使用，用户不可以使用。

3. 命令表

一个程序可以通过在系统输入库中生成一个名为 XXXXCMDS 的表，这个表被用来当做此应用程序的命令表。XXXX 为应用程序的 ID，为 1～4 位的字符。如 ISPCMDS 是 ISPF 分配的系统命令表。

此外，用户可以指定最多 3 个用户命令表和 3 个 SITE 命令表，这些配置信息记录在 ISPF 配置表中。用户还可以指定 SITE 命令表是在系统命令表之前还是之后被检索。

定义一个应用程序命令表，可以使用对话框标记语言（DTL）和 ISPF 的 conversion 实用程序，或者使用 ISPF 的选项 3.9 来完成，如图 7-5 所示。

```
  Menu  Help
- ---------------------------- Commands ----------------------------------
                           Command Table Utility

1    Specifications                          Command table search order
     Application ID . . ISP                  Application table  . : ISP
2    Enter "/" to select option              User table 1 . . . . :
        Show description field               User table 2 . . . . :
3                                            User table 3 . . . . :
4                                            Site table 1 . . . . :
                                             Site table 2 . . . . :
5                                            Site table 3 . . . . :
6                                            System table . . . . : ISP
7
8  If no application ID is specified, the current application ID will be
9  used. The name of the command table to be processed is formed by
1  prefixing the application id to the string 'CMDS'.  For example:
1  Application ID  . .  TST results in a command table name of 'TSTCMDS'.
1
1  Command ===>
1   F1=Help       F2=Split      F3=Exit       F7=Backward   F8=Forward
0   F9=Swap      F12=Cancel
 --------------------------------------------------------------------------
F10=Actions  F12=Cancel
```

图 7-5　ISPF 选项 3.9 定义程序命令表

按回车键，则进入想要查看的命令表显示窗口，如图 7-6 所示。

```
  Menu  Utilities  Help

                             Display ISPCMDS             Row 1 to 12 of 88

The command table is currently open, it cannot be modified. Use the view(V)
row command to see an entire entry.

     Verb      T  Action
     BACKWARD  0  ALIAS UP
     BOTTOM    0  ALIAS DOWN MAX
     FORWARD   0  ALIAS DOWN
     TOP       0  ALIAS UP MAX
     ACTIONS   0  ACTIONS
     AUTOTYPE  0  SETVERB
     CANCEL    3  CANCEL
     CMDE      0  SELECT PGM(ISPCCMDE) SUSPEND
     COLOR     0  SELECT PGM(ISPOPT) PARM(ISPOPT10) SCRNAME(SETTINGS)
     CRETRIEV  0  CRETRIEV
     CUAATTR   0  SELECT PGM(ISPOPT) PARM(ISPOPT11) SCRNAME(SETTINGS)
     CURSOR    0  CURSOR

Command ===>                                                Scroll ===> PAGE
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

图 7-6　命令显示表

上例中显示了系统正在使用的命令表，可在任意条命令前面输入 V 来显示命令的具体描述。

用户在命令输入位置输入相应命令时，系统会立即做出反应。如在命令输入位置输入 ACTIONS，则光标会移到 action bar。

当用户输入一个命令，对话框管理器搜索应用程序命令表，然后搜索系统命令表。如果找到了命令，立即执行；如果没有找到命令，这条命令则被传回对话界面，原样地显示出来，由对话界面采取相应的反馈信息给用户。

4. 面板定义

一个面板的定义是面板的编程描述，它定义了面板的内容和格式。大多数面板提示用户输入，输入的信息被记录为数据，传递到“数据输入面板”。在“选择面板”中，通过对话程序系统可以识别用户的输入是哪一条路径。面板可以调用 REXX 语句，执行诸如算术运算、数据转换、对话变量的格式化、验证、转换等功能。

5. 消息定义

消息定义指定了消息的形式和文本内容。消息可以确认用户输入的请求是正在执行还是已完成，或者报告用户的输入错误。消息可以显示定义的信息，或者可以复制日志，或者采用二者结合显示信息。

总结一下，参照图 7-7，开发人员使用 ISPF 服务创建和测试对话框元素。面板的定义、消息定义和文件裁剪框架等的创建需要在运行对话框之前完成，这些对话框元素被

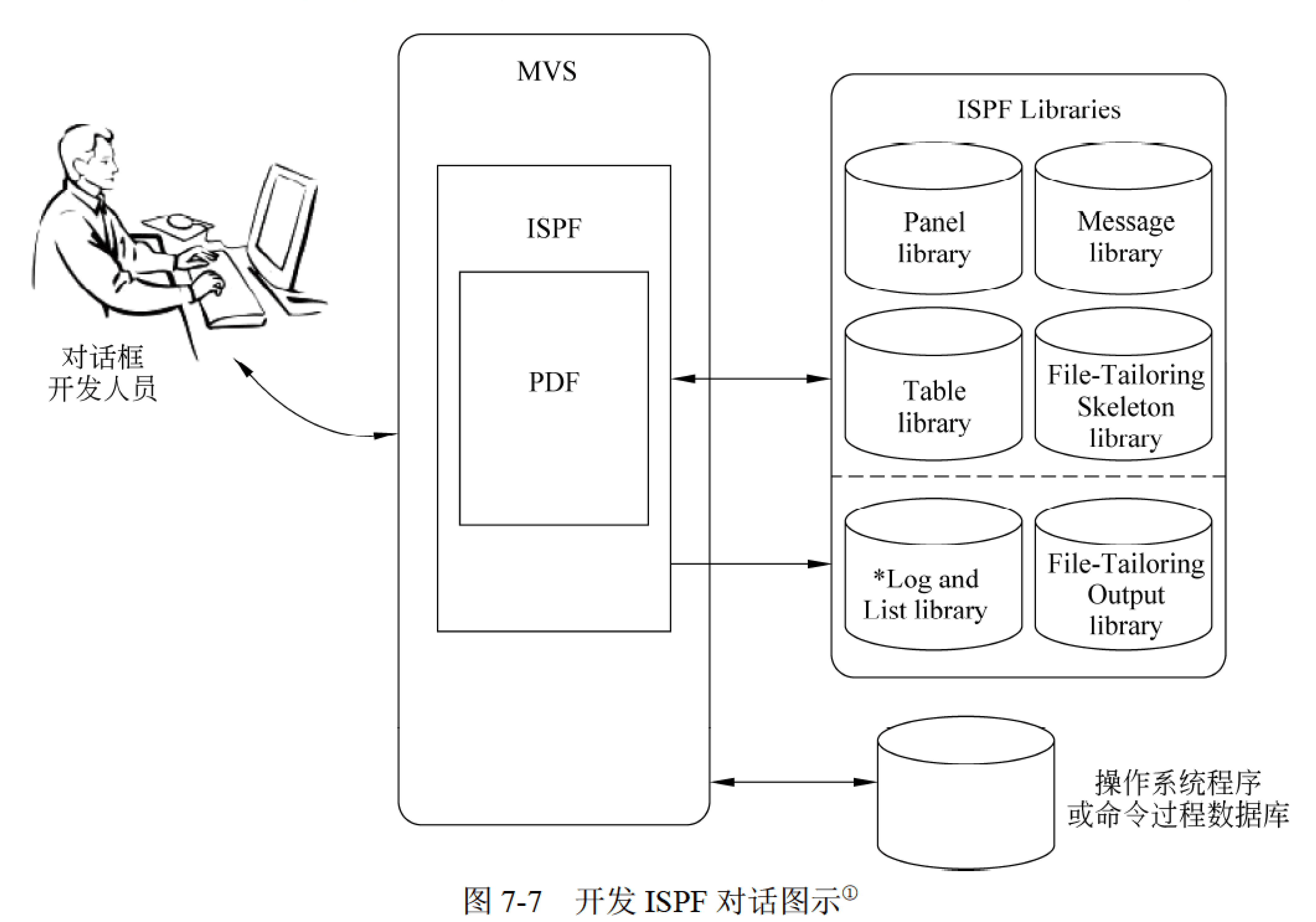


图 7-7　开发 ISPF 对话图示①

① 摘自 IBM 白皮书 *z/OS V1R6.0 ISPF Dialog Developer's Guide and Reference* 图 5。

保存在库中。开发人员把程序（编译后）或命令过程存储到一个适当的系统程序库中。在对话测试过程中，对话框处理过程生成了数据表、日志记录和文件裁剪输出数据集。ISPF 将在用户第一次执行 ISPF 操作时生成日志数据集，之后 ISPF 将用户操作的结果作为日志信息进行记录。

当开发人员完成了功能函数、面板的定义以及应用程序需要的任何其他对话元素后，对话就可以在 ISPF 下运行。更多关于对话元素的介绍，请参阅 IBM 白皮书 *ISPF Services Guide*。

7.2 ISPF 服务调用

除了基于图形面板的界面外，ISPF 提供应用程序的交互接口——ISPF 服务。通过 ISPF 服务交互接口，可以进行交互式的 ISPF 对话应用开发，或者在程序中使用 ISPF 提供的各种功能（例如数据集浏览、编辑）、在后台启动 ISPF 对话（显示一个对话面板）等各种功能。

程序员可以使用命令或者程序中的 CALL 语句来调用 ISPF 服务。事实上，ISPF 服务是 ISPF 提供给各种编程语言的一个通用的交互接口，提供了 100 多个 ISPF 服务。每种语言都可依照各自语法规进行 ISPF 服务的调用，这些语言包括 REXX、CLIST、PL/I、COBOL、C，等等。这里主要介绍在 REXX 中如何调用 ISPF 服务。

通常来说，在使用 ISPF 服务之前，需要在程序中首先检测 ISPF 服务是否可用，可以通过 ISPQRY 命令进行检测，其返回码 RC 结果有两种，即 0 表示服务对当前调用者可用；20 表示对当前调用者不可用。例如：

```
/* REXX */
ADDRESS TSO
"CALL 'ISP.SISPLOAD(ISPQRY)'"
SAY 'Return Code='RC
```

调用 ISPF 服务有两种形式，即使用命令调用 ISPF 服务和使用程序调用 ISPF 服务。二者的不同主要在于调用的形式。编程语言如 COBOL 和 FORTRAN 等可以使用程序调用方式，REXX 则常采用命令调用方式。

7.2.1 使用命令调用 ISPF 服务

使用命令来调用 ISPF 服务又可以使用以下的两种方式。

（1）在 REXX 程序中，调用 ISPEXEC 命令调用服务命令。

（2）使用 ISPF 面板选项 7.6：即 Dialog Test 选项面板的 Dialog Services 选项。

使用 ISPEXEC 命令接口调用 ISPF 服务是 REXX 程序中最为常用的方法，通常，在 REXX 中调用 ISPEXEC 接口调用的一般形式如下：

```
ISPEXEC service-name parameter1 parameter2 parameter3 …
```

例如：

```
ISPEXEC DISPLAY PANEL(XYZ)
```

其中：parameter1 常为某些服务需要的位置参数；parameter2、parameter3 等通常为以下两种形式之一：keyword 或者 keyword (value)。调用语句中 service-name 必须为大写字母。service-name 和 parameter 也可以使用 REXX 中的变量，使用&前缀。

但有一些服务不能使用 ISPEXEC 直接调用，这些服务如下。

GRERROR，VCOPY，VMASK

GRINIT，VDEFINE，VREPLACE

GRTERM，VDELETE，VRESET

以上服务可以使用程序中的 CALL 语句来调用。

在命令过程语句的函数中，通过 ISPEXEC 命令调用 ISPF 服务。例如：

```
ISPEXEC DISPLAY PANEL(XYZ)
```

上例在当前的终端上显示 XYZ 面板，XYZ 面板的定义预先要存储在面板定义文件中，指定面板上的内容及其显示的形式。除了上面直接使用 ISPEXEC 接口之外，也可以首先在 REXX 程序中转换到 ISPEXEC 宿主环境，然后调用 ISPF 服务。例如：

例 1：显示 ISPF 面板和弹出对话框。

```
/* REXX */
ADDRESS ISPEXEC /* 转换宿主环境 */
   "ADDPOP"
   "DISPLAY PANEL(PANELA)"
   "ADDPOP POPLOC(FIELD2)"
      ZWINTTL = "POPUP WINDOW TITLE"
   "DISPLAY PANEL(PANELB)"
   "ADDPOP COLUMN(5) ROW(3)"
      ZWINTTL = ""
   "DISPLAY PANEL(PANELC)"
EXIT
```

程序运行结果如图 7-8 所示。

例 2：数据集处理。

本例给出了通过在 REXX 程序中调用“库访问服务”来删除某一个分区数据集下的某成员的方法。在调用“库访问服务”时，需要使用 LMDINIT 将输入的数据集名称 DSNAME 同 DATAID 关联起来，从而使别的服务可以通过该 ID 定位该分区数据集。程序执行完成后，需要释放 DATAID 与数据集的关联。

```
/********************REXX**********************/
PULL DSNAME MBNAME
/*** DSNAME 是数据集名称,MBNAME 是成员名称,需要用户从屏幕上输入 ***/
   "SUBCOM ISPEXEC"
```

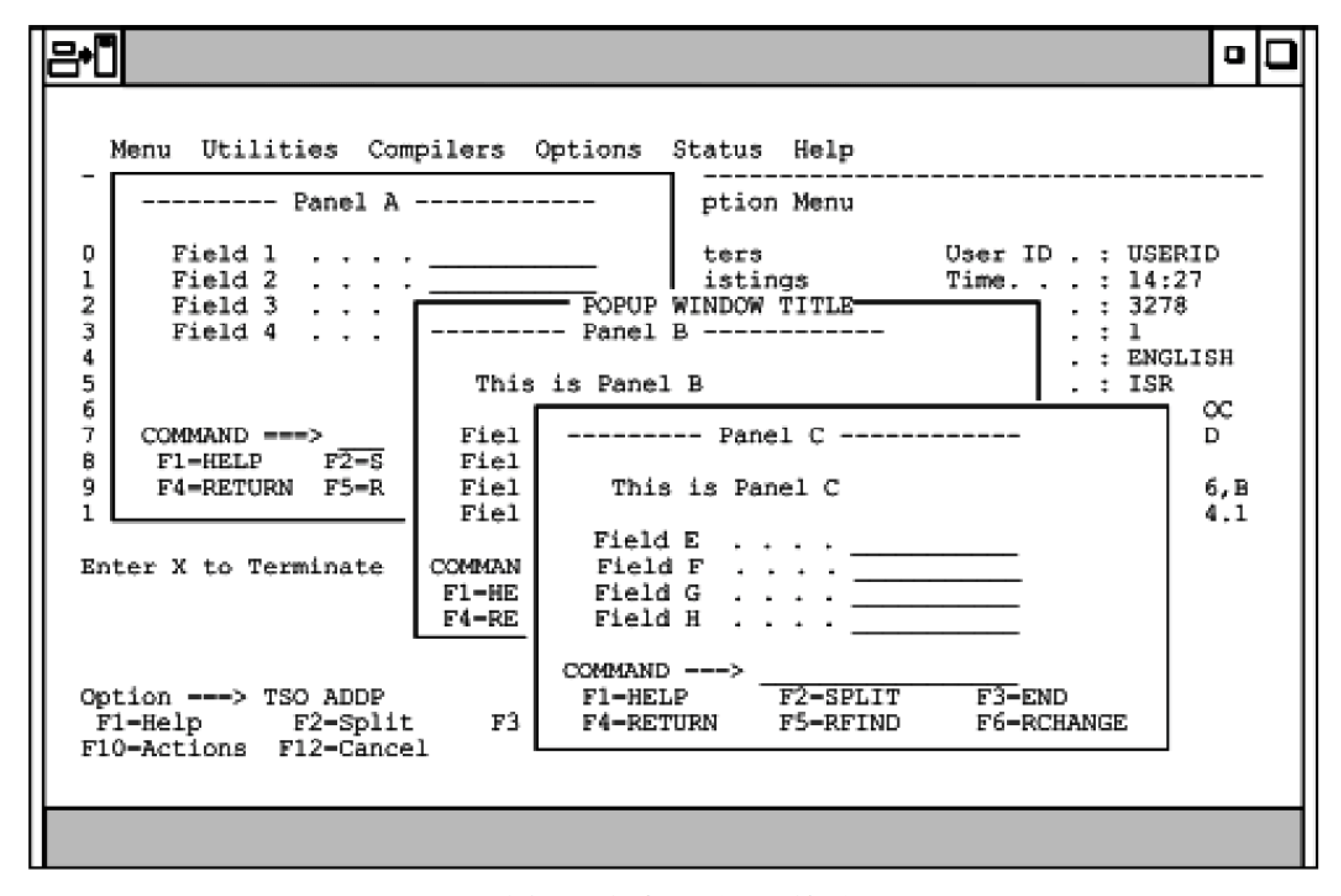


图 7-8 程序调用显示结果

```
    ADDRESS ISPEXEC
      'LMINIT DATAID('DDVAR') DATASET('DSNAME') ENQ(SHRW)'
      /*** 赋给数据集 DSNAME 一个 DATAID: DDVAR ***/
      SAY 'The ID OF the Data Set:' DSNAME 'is' DDVAR
      'LMOPEN DATAID('DDVAR') OPTION(OUTPUT) LRECL('DLVAR')
      RECFM('RFVAR') ORG('ORGVAR')'
      /*** 将数据集名称与 DATAID 关联起来 ***/
      'LMMDEL DATAID('DDVAR') MEMBER('MBNAME')'
      /*** 删除该数据集下的成员 MBNAME ***/
      'LMFREE DATAID('DDVAR')' /*** 释放 DATA-ID 与数据集关联 ***/
   IF (RC <> 0) THEN
      SAY 'ERROR HAPPEN'
EXIT
```

通过上例可以看到，通过这种方法可以调用 ISPF 所提供的各种文件操作，例如 LMINIT （创建 DATAID）、LMRENAME（重命名数据集）以及 LMGET（读取数据集中的一条记录）。大大丰富了 REXX 脚本程序的自身功能。

7.2.2 传递 Dialog 变量作为参数

一些 ISPF 服务允许使用 Dialog 变量作为参数传递给命令语句，变量名称不能超过 8 个字符。这些 Dialog 变量不必使用前缀&，除非想要将它们换成其他的变量。例如：

```
ISPEXEC  VGET  XYZ
ISPEXEC  VGET  &VNAME;
```

VGET 是复制变量内容的语句，在第一个语句中，XYZ 就是传递的 Dialog 变量。在第二个语句中，变量 VNAME 则包含了要被传递的 Dialog 变量。

VGET 还可以把通过一个参数获取一系列变量名。VGET 服务的语法如下。

```
ISPEXEC  VGET  name-list  [ASIS|SHARED|PROFILE]
```

其中 name-list 是一个位置参数。它能够包括多达 254 个 Dialog 变量名，每一个变量名用逗号或者空格隔开。当 name-list 包括多个变量时，需要用括号。如果只有一个变量时，则可以省略括号，VGET 获取一系列 Dialog 变量的例子如下。

```
ISPEXEC  VGET  (AAA,BBB,CCC)
ISPEXEC  VGET  (LNAME FNAME I)
ISPEXEC  VGET  (XYZ)
ISPEXEC  VGET  XYZ
```

7.2.3　ISPF 编辑器

ISPF 编辑器也提供了对应的接口服务。通过这些接口的使用，可以使程序运行于 ISPF 下，并能够使用 ISPF 以及 ISPF 编辑器提供的各种功能，例如编辑命令操作（如 Find/Change 等）、数据操作（如 Shift/Sort 等）、行操作（如 Move/Insert 等）以及基本文件操作（Save/Submit 等）以及运行编辑宏等近百种服务，都可以通过 ISPF 编辑器服务接口提供。和前面所提到的“ISPF 服务”类似，ISPF 编辑器服务运行所需要宿主环境是 ISREDIT，通过 REXX 的 ADDRESS 指令语句切换环境：Address ISREDIT。

可在 REXX 程序中使用编辑器的 RESET 命令，命令如下：

```
ADDRESS ISREDIT RESET
```

IPSF 编辑器服务的用法和前面所述的 ISPEXEC 宿主环境中 ISPF 服务的语法相同。ISPF 编辑器服务在编辑宏(Edit Macro)中有广泛应用。

7.2.4　调用 ISPF 服务的返回值

通常 ISPF 服务调用结束后，会有对应的命令返回码，通过特殊变量 RC 变量可以查询，这些返回码的含义如表 7-1 所示，更多关于返回码的详细信息，参阅 IBM 白皮书 *ISPF Services Guide*。

表 7-1　ISP 服务调用返回码

返 回 码	含　义	说　　明
0	正常完成	服务正确完成返回
4，8	异常产生	警告信息，不一定为错误产生。当返回码为 4 时，表示有一些警告信息。当返回码为 8 时，表示有非严重的错误产生，例如到达数据集或成员列表末尾

续表

返 回 码	含　义	说　　明
10，12，14，16，20	错误产生	服务产生错误，未能正常完成。当返回码为 12 或更大时，可使用 CONTROL 服务来控制错误。10 和 14 是 PDF（Program Development Facility）组件服务特有的返回码

7.3　ISPF 服务描述

7.3.1　ISPF 服务分类

ISPF 服务分为 8 类：对话显示服务、文件裁剪服务、库访问服务、PDF 组件服务、表格服务、变量服务、杂项服务。

1. 对话显示服务

该类服务包括了诸如显示/消除 ISPF 弹出窗口（ADDPOP/REMPOP）、显示面板信息（DISPLAY/SELECT）、信息控制（SETMSG/TBDISPL）等对话控制服务。该类服务汇总如表 7-2 所示。

表 7-2　对话显示服务汇总

服　　务	说　　明
ADDPOP	指定显示窗口顺序和位置
DISPLAY	从窗口文件中读取面板的定义、初始化变量信息，并在屏幕上显示此窗口
REMPOP	从屏幕中移除置顶窗口
SELECT	用来显示可选择窗口的层次结构或者调用一个函数
SETMSG	从 ISPF 系统预留区域的信息文件中读取并构建一个特定的信息。此信息将在下一个窗口使用任意 DM 组件服务进行中显示
TBDISPL	从存储窗口定义信息的 ISPF 表中读取并组合信息。显示从表中选取的行允许用户进行处理

2. 文件裁剪服务

文件裁剪服务包括了对文件进行裁减操作的各种服务，例如：FTOPEN、FTINCL、FTCLOSE、FTERASE 等。文件裁剪服务，以正常被调用的顺序来列出。该类服务汇总如表 7-3 所示。

表 7-3　文件裁剪服务

服　　务	说　　明
FTOPEN	准备文件裁剪处理，指定是否把输出放到临时文件中
FTINCL	指定使用的框架，并开始裁剪处理
FTCLOSE	结束文件裁剪处理
FTERASE	删掉文件裁剪过程产生的输出文件

3. 库访问服务

库访问服务提供了丰富的访问和操作 ISPF 库以及文件的各种功能服务，例如复制一个分区数据集成员，实现的功能和 ISPF 的 3.4、3.3 以及 3.2 接口相似。该类服务汇总如表 7-4 所示。

表 7-4　库访问服务汇总

服　务	说　明
DSINFO	在窗口变量中显示一个分区数据集的信息
LMCLOSE	关闭一个数据集
LMCOMP	使用新的 compress request exit 压缩一个分区数据集，如果 exit 没有安装，则使用 IEBCOPY 来进行压缩
LMCOPY	复制分区数据集成员或顺序数据集，允许打包、加锁和截断操作
LMDDISP	显示一个指定数据集列表 ID 的数据集列表
LMDFREE	移除一个数据集 ID 与一个 DSNAME LEVEL 和 VOLUME combination 的连接
LMDINIT	把一个数据集 ID 与一个 DSNAME LEVEL 和 VOLUME combination 相连接。之后，这个数据集列表 ID 在其他库通过服务来处理时被用来标识 DSNAME LEVEL 和 VOLUME combination
LMDLIST	生成一个指定数据集列表 ID 的数据集
LMERASE	删除 ISPF 库或者 MVS 数据集
LMFREE	释放一个同指定 DATA-ID 连接的数据集
LMGET	读取一个数据集的一条记录
LMINIT	把一个或多个 ISPF 库或者以存在的数据集同一个 DATA-ID 连接起来。之后，这个 DATA-ID 在其他库通过服务来处理时被用来标识数据集
LMMADD	添加一个成员到一个 ISPF 库或者一个分区数据集
LMMDEL	删除一个 ISPF 库或者一个分区数据集的成员
LMMDISP	为下面的数据集提供成员选择列表
LMMFIND	查找一个 ISPF 库或者一个分区数据集的成员
LMMLIST	生成一个 ISPF 库或者一个分区数据集的成员列表
LMMOVE	删除分区数据集成员或者顺序数据集，允许打包和自动截取选项
LMMREN	重命名 ISPF 库或者分区数据集的一个成员
LMMREP	替换 ISPF 库或者分区数据集的一个成员
LMMSTATS	对含有固定长度或者可变长度记录的分区数据集成员，生成、存储或者删除 ISPF 的统计信息
LMOPEN	打开一个数据集
LMPRINT	用可选的固定格式，来打印数据集列表
LMPUT	写一个数据集的一条记录
LMQUERY	提供与一个指定 DATA-ID 相连接的数据集的信息
LMRENAME	重命名一个 ISPF 库

4. PDF 组件服务

PDF 组件服务包括 BRIF（浏览接口）、BROWSE、EDIT（编辑接口）、EDIREC（编辑恢复）、EDIT、VIEW、VIIF 和 EDREC（对 EDIT 和 VIEW 编辑复原）等。该类服务汇总如表 7-5 所示。

表 7-5 PDF 组件服务汇总

服 务	说 明
BRIF	利用窗口提供的 I/O 流程提供数据浏览功能。允许浏览分区数据集和顺序数据集，例如子系统数据和 IN-STORAGE 数据。而且在浏览之前会进行数据的预处理
BROWSE	用来查看任意的 ISPF 库，或者可以用 LMINIT 服务来生成的数据集，和一些 ISPF 不支持的其他数据类型。可以浏览工作站和工作站文件的主数据集
EDIF	利用窗口提供的 I/O 流程提供数据编辑功能。允许编辑分区数据集和顺序数据集，例如子系统数据和 IN-STORAGE 数据。而且在编辑之前会进行数据的预处理
EDIREC	用 EDIF 服务初始化一个编辑复原表(ISREIRT),终止用 EDIF 服务来复原时的 PENDING 状态
EDIT	用来查看 ISPF 库，和 ISPF 库的集合，或者可以用 LMINIT 服务来生成的数据集。EDIT 服务提供一个对 PDF 编辑器的接口，跳过显示 Edit Entry Panel。也可以编辑工作站和工作站文件的主数据集
EDREC	初始化一个编辑或查看复原表。终止复原时的 PENDING 状态，执行第一个声明的动作
VIEW	功能同 EDIT 服务相似，但有下面的不同： ● 必须使用 REPLACE 或者 CREATE 主命令来保存数据 ● 当在 VIEW 模式下更改文件后，输入 END 主命令，将提示是否保存更改
VIIF	利用窗口提供的 I/O 流程提供数据查看（VIEW）功能。允许查看分区数据集和顺序数据集，例如子系统数据和 IN-STORAGE 数据。而且在查看之前会进行数据的预处理

5. 表格服务

表格服务提供了表格操作的各种服务，分为表级服务和行级服务。该类服务汇总如表 7-6 和表 7-7 所示。

表 7-6 影响整个表的服务汇总

服 务	说 明
TBCLOSE	关闭表，假如表被打开则保存一个长期的备份
TBCREATE	生成并开打一个新表用作处理
TBEND	关闭表不保存更改
TBERASE	从表输入文件删除一个长期表
TBOPEN	打开一个存在的长期表用作处理
TBQUERY	获取一个表的信息
TBSAVE	在打开装填保存一个表的永久备份
TBSORT	对表做排序
TBSTATS	打开一个表的统计信息

表 7-7　影响表中行的服务汇总

服　　务	说　　明
TBADD	在表中添加一行
TBBOTTOM	在最后一行添加 CRP，并恢复行
TBDELETE	从表中删除一行
TBEXIST	利用键值确认是否存在某行
TBGET	恢复表中的一行
TBMOD	更新表中的一行，或者添加一个新的行
TBPUT	如果存在或者键值匹配则更新行
TBSARG	同 TBSCAN 或 TBDISPL 一起建立一个检索概要（search argument）
TBSCAN	检索行匹配一组变量的表
TBSKIP	向前或向后移动 CRP 指定的行数，然后把 CRP 所在行 retrieves
TBTOP	将 CRP 置顶（置于第一行起始位置）
TBVCLEAR	把对应表中的变量的对话框变量设置为 NULL

6. 变量服务

变量服务提供了传递和操作 ISPF 变量的各种服务。如前例所提到的 VGET。该类服务汇总如表 7-8 所示。

表 7-8　可在所有功能中使用的变量服务

服　　务	说　　明
VERASE	从 SHARED POOL 和/或 PROFILE POOL 移除变量
VGET	从 SHARED POOL 或 PROFILE POOL 恢复变量
VPUT	仅更新 SHARED POOL 或 PROFILE POOL 中的变量

表 7-9　仅在程序中可以使用的变量服务

服　　务	说　　明
VCOPY	从一个对话框变量复制数据到程序中
VDEFINE	在 ISPF 中定义功能程序变量
VDELETE	移除功能变量的定义
VMASK	把一个 MASK 赋给一个对话框变量
VREPLACE	用程序中的数据更新对话框变量
VRESET	重置功能变量

7. 杂项服务

杂项服务是 ISPF 所提供的其他有用服务，如可以配置日志登录的 LOG 服务。该类服务汇总如表 7-10 所示。

表 7-10　杂项服务汇总

服　务	说　明
CONTROL	允许一个功能来控制 ISPF 显示特定形式的输出，或者控制对话管理服务中遇到的错误处置
FILESTAT	在连接的工作站上提供有关网站的统计数据
FILEXFER	向工作站上传文件或从工作站下载文件
GETMSG	获得消息和相关信息，并把他们存储在服务中指定的变量中
GRERROR	提供访问 GDDM 错误记录的地址和 GDDM 调用格式描述符模块地址
GRINIT	初始化 ISPF/ GDDM 接口和可选的请求，ISPF 面板定义一个 GDDM 图形领域
GRTERM	终止以前建立的 GDDM 接口
LIBDEF	在一个活跃的 ISPF 对话中，提供一种动态定义应用程序数据元素文件的方法
LIST	允许一个对话框直接将数据线写入（不使用打印命令或 UTILITIES）ISPF 的列表数据集
LOG	允许一个函数向 ISPF 的日志文件中写入一条消息。用户可以指定当 ISPF 终止时日志是否要打印、保存或删除
PQUERY	返回为一个特定的面板上的一个特定区域的信息。在变量中返回与该区域相关特性的类型、大小和位置
QBASELIB	使 ISPF 对话框获得指定 DDNAME 的当前库的信息
QLIBDEF	允许 ISPF 对话框，向获取当前 LIBDEF 定义信息，这些信息可以被对话框保存，用于以后恢复任何可能被覆盖的 LIBDEF 定义
QTABOPEN	允许 ISPF 对话框中获得当前打开的 ISPF 的表的列表。 TBSTATS 或 TBQUERY 服务，然后可以用来获取关于每个表的更详细的信息
QUERYENQ	允许 ISPF 对话框，获得了所有系统排入队列列表，或指定了条件匹配的所有系统排入队列的列表
TRANS	从一个编码字符集标识符（CCSID）传递数据到另一个
WSCON	使用户连接到工作站，而无须使用 ISPSTART 命令的 GUI 参数，并从 ISPF 设置中启动工作站连接面板
WSDISCON	允许用户从工作站断开，而不必终止 ISPF 对话

7.3.2　几个常用的 ISPF 服务

1. ADDPOP——开启弹出窗口模式

ADDPOP 服务的功能是把随后会出现的所有弹出窗口都显示在一个窗口中，来提示对话框管理者随后会出现的弹出窗口。查看 ADDPOP 的所有窗口需调用 DISPLAY、TBDISPL 或者 SELECT PANEL 服务。

使用 ADDPOP 服务时，每一个弹出窗口可以有设置一个窗口的标题。标题是嵌在窗框边框的顶部，最大只能有一个边线的长度。如果标题长于窗口边线，管理器会将其截断。定义窗口的标题，是通过设置系统变量 ZWINTTL。

命令调用格式：

```
ISPEXEC ADDPOP [POPLOC(field-name)]
               [ROW(row)]
               [COLUMN(column)]
```

2. LIBDEF——定义程序库

LIBDEF 服务提供了应用程序数据集的动态定义，从而使应用程序的数据集可以在 ISPF 的对话中使用。这就免除了在调用 ISPF 对话之前要用分配的语句来定义所有的应用数据集的烦琐操作。

LIBDEF 服务可以被用来定义应用程序级的以下库：

- 面板库；
- 消息库；
- 表库；
- 框架库；
- 文件裁剪输出库；
- 用户连接库；
- 图像库。

用来定义 ISPF 的库的 DDNAMES，同样用来定义数据集上的 LIBDEF 服务请求。ISPPROF 和 ISPF 的配置文件库在应用程序级的定义是不允许的，因为 ISPPROF 包含用户相关的数据。

LIBDEF 服务提供了 4 种方式来定义应用程序级的数据集：DATASET、EXCLDATA、LIBRARY、EXCLLIBR。

命令调用格式：

```
ISPEXEC LIBDEF lib-type [DATASET|EXCLDATA|LIBRARY|EXCLLIBR]
                        [ID(dataset-list)|ID(libname)]
                        [COND|UNCOND|STACK|STKADD]
```

示例 1：DATASET 关键字

在进入 ISPF 之前，用户发出下面的 ALLOCATE 语句来定制一个面板库。

```
ALLOCATE DATASET('ISPFPROJ.ABC.MYPAN') FILE(ISPPUSR) SHR
ALLOCATE DATASET('ISPFPROJ.ABC.PANELS') FILE(ISPPLIB) SHR
```

接下来，DATASET 关键字设置 LIBDEF 服务来定义一个应用程序级的面板库（分区数据集）。

```
ISPEXEC LIBDEF ISPPLIB DATASET ID('ISPFPROJ.ABC.APPAN1',
'ISPFPROJ.ABC.APPAN2')
```

示例 2：EXCLDATA 关键字

首先在进入 SIPF 之前定义用户连接库。

```
ALLOCATE DATASET('ISPFPROJ.ABC.MYMOD') FILE(ISPLUSR) SHR
ALLOCATE DATASET('ISPFPROJ.ABC.LLOAD') FILE(ISPLLIB) SHR
```

接下来，用 EXCLDATA 关键字设置 LIBDEF 服务来定义一个应用程序级的连接库（分区数据集）。

```
ISPEXEC LIBDEF ISPLLIB EXCLDATA ID('ISPFPROJ.ABC.APMOD1',
                                   'ISPFPROJ.ABC.APMOD2')
```

示例 3：LIBRARY 关键字

首先在进入 ISPF 之前定义用户连接库。

```
ALLOCATE DATASET('ISPFPROJ.ABC.APPAN1',
                 'ISPFPROJ.ABC.APPAN2') FILE(APPLIB) SHR
ALLOCATE DATASET('ISPFPROJ.ABC.MYPAN') FILE(ISPPUSR) SHR
ALLOCATE DATASET('ISPFPROJ.ABC.PANELS') FILE(ISPPLIB) SHR
```

接下来，用 LIBRARY 关键字设置 LIBDEF 服务来定义一个应用程序级面板库。

```
ISPEXEC LIBDEF ISPPLIB LIBRARY ID(APPLIB)
```

示例 4：EXCLLIBR 关键字

首先在进入 ISPF 之前定义用户连接库。

```
ALLOCATE DATASET('ISPFPROJ.ABC.APMOD1',
                 'ISPFPROJ.ABC.APMOD2') FILE(APLLIB) SHR
ALLOCATE DATASET('ISPFPROJ.ABC.MYMOD') FILE(ISPLUSR) SHR
ALLOCATE DATASET('ISPFPROJ.ABC.LLOAD') FILE(ISPLLIB) SHR
```

接下来，用 EXCLLIBR 关键字设置 LIBDEF 服务来定义一个应用程序级的用户连接库。

```
ISPEXEC LIBDEF ISPLLIB EXCLLIBR ID(APLLIB)
```

3. EDIT——编辑数据集

EDIT 服务提供了对 PDF 编辑器的接口，绕过了对 the Edit Entry Panel 的显示。EDIT 接口允许使用自己定义的面板来显示数据，可以指定初始宏和配置文件。也可以是用 EDIT 来查看 ISPF 库，以及使用 LMINIT 服务生成的数据集。可以递归式地使用 EDIT 服务，比如当处在编辑窗口时，再输入 EDIT 命令。

当在恢复模式（recovery mode）中使用 EDIT 服务，所做的所有操作会自动地记录在一个 PDF 控制数据集中。当发生错误，用户就可以使用控制数据集中的内容来复原用户正在编辑的数据集。

命令调用格式如下：

```
ISPEXEC  EDIT  DATASET(dsname)  [VOLUME(serial)]
                                [PASSWORD(pswd-value)]
                                [PANEL(panel-name)]
                                [MACRO(macro-name)]
                                [PROFILE(profile-name)]
```

```
                                          [FORMAT(format-name)]
                                          [MIXED(YES|NO)]
                                          [LOCK(YES|NO)]
                                          [CONFIRM(YES|NO)]
                                          [WS(YES|NO)]
                                          [PRESERVE]
                                          [PARM(parm-var)]
OR
   ISPEXEC  EDIT  DATAID(dsname)   [MEMBER(member-name)]
   ⋮
OR
   ISPEXEC EDIT WSFN(ws-filename)  [ PANEL(panel-name)]
   ⋮
```

示例为 TELOUT 服务来调用 EDIT 服务，编辑数据集 ISPFPROJ.FTOUTPUT 的一个成员。

命令调用：

```
ISPEXEC EDIT DATASET('ISPFPROJ.FTOUTPUT(TELOUT)') WS(YES)
```

或者

```
ISPEXEC LMINIT DATAID(EDT) DATASET('ISPFPROJ.FTOUTPUT')
ISPEXEC EDIT DATAID(&EDT) MEMBER(TELOUT) WS(YES)
```

4. LMCOPY——复制一个数据集的成员

LMCOPY 服务可以复制一个分区数据集的成员或者一个完整的顺序数据集。可以选择性地进行打包数据、对成员加锁、替换成员或者自动截断。但只能对记录是定长（fixed-record）或变长（variable- record）的数据集进行打包操作。

命令调用格式：

```
ISPEXEC LMCOPY FROMID(from-data-id)
                        [FROMMEM(from-member-name)]
                        TODATAID(to-data-id)
                        [TOMEM(to-member-name)]
                        [REPLACE]
                        [PACK]
                        [TRUNC]
                        [LOCK]
                        [SCLMSET(Y|N)]
                        [ALIAS|NOALIAS]
```

示例：调用 LMCOPY 服务来复制所有字母“L”开头的成员，所在的数据集的 data ID 为变量 DDVAR，目标数据集的 data ID 为变量 DDVAR2。同样名字的成员会被替换。同时可以使用打包和截断操作。

命令调用：

```
ISPEXEC LMCOPY FROMID(&DDVAR) FROMMEM(L*) +
               TODATAID(&DDVAR2) REPLACE PACK TRUNC
```

更多的 ISPF 服务描述，可以参阅 IBM 白皮书 *ISPF Service Guide*。

7.4 ISPF 对话设计架构

7.4.1 控制流和数据流

本节介绍怎样去控制一个 ISPF 会话，包括开始和结束一个会话以及对 ISPF 的使用。

图 7-9 显示了对话控制和数据流。在一个 ISPF 会话开始，可以使用 ISPSTART 命令或者使用 SELECT 选择面板来调用一个对话功能。图中也显示了 ISPF 服务同各个对话元素的交互。

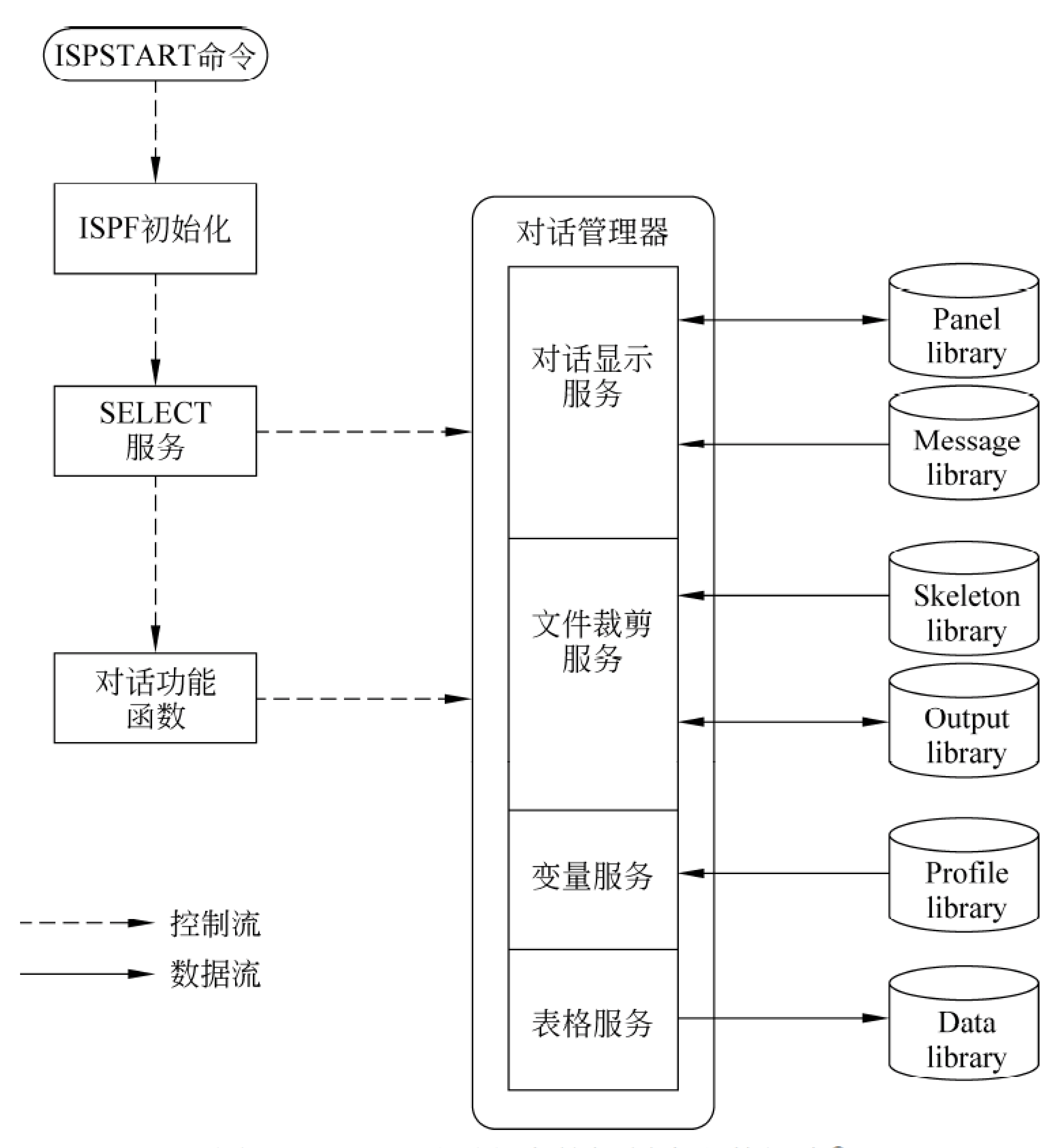


图 7-9　ISPF 对话程序的控制流和数据流[①]

7.4.2 对话组织方式

ISPF 对话可以有多种组织方式来满足用户的需求，下面介绍两种常用的方法。

① 摘自 IBM 白皮书 *z/OS V1R6.0 ISPF Dialog Developer's Guide and Reference* 中图 8。

1. 经典组织方式一

一个典型的对话组织方式在图 7-10 给出。对话流程从上至下，开始于调用主选项菜单，用户在主选项菜单中做出选择来调用一个功能或者显示下一个下层菜单。每一个下层菜单同样可以启动一个功能或者显示另一个下层菜单。

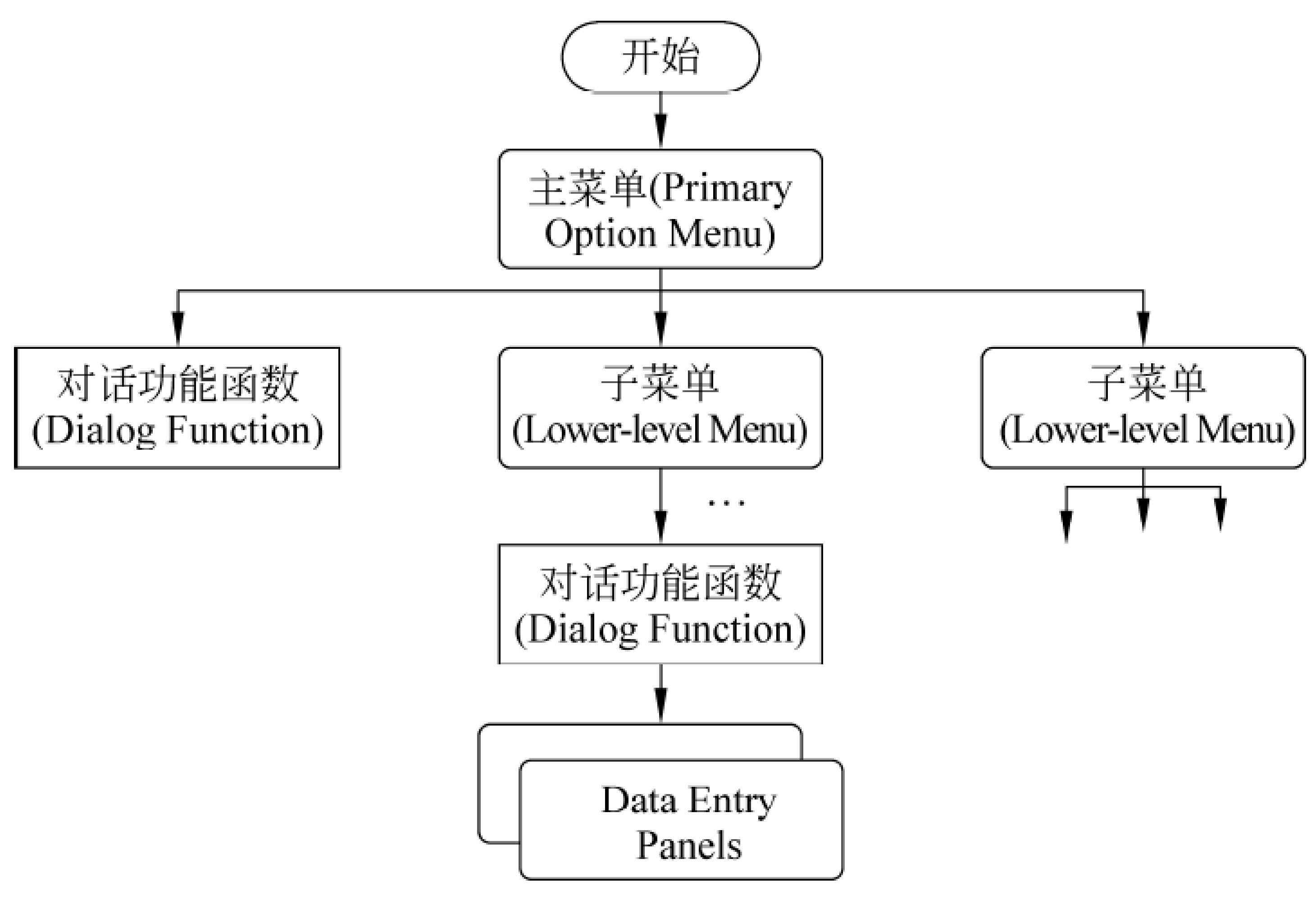


图 7-10　对话框组织方式 1①

2. 经典组织方式二

图 7-11 显示了另外一种窗口组织方式，首先调用了一个对话的功能函数，这个功能函数执行程序的初始化，通过数据输入面板（data-entry panels）来提示基本信息给用户，然后应用程序通过调用 SELECT 服务来启动选择处理过程显示主选项菜单。

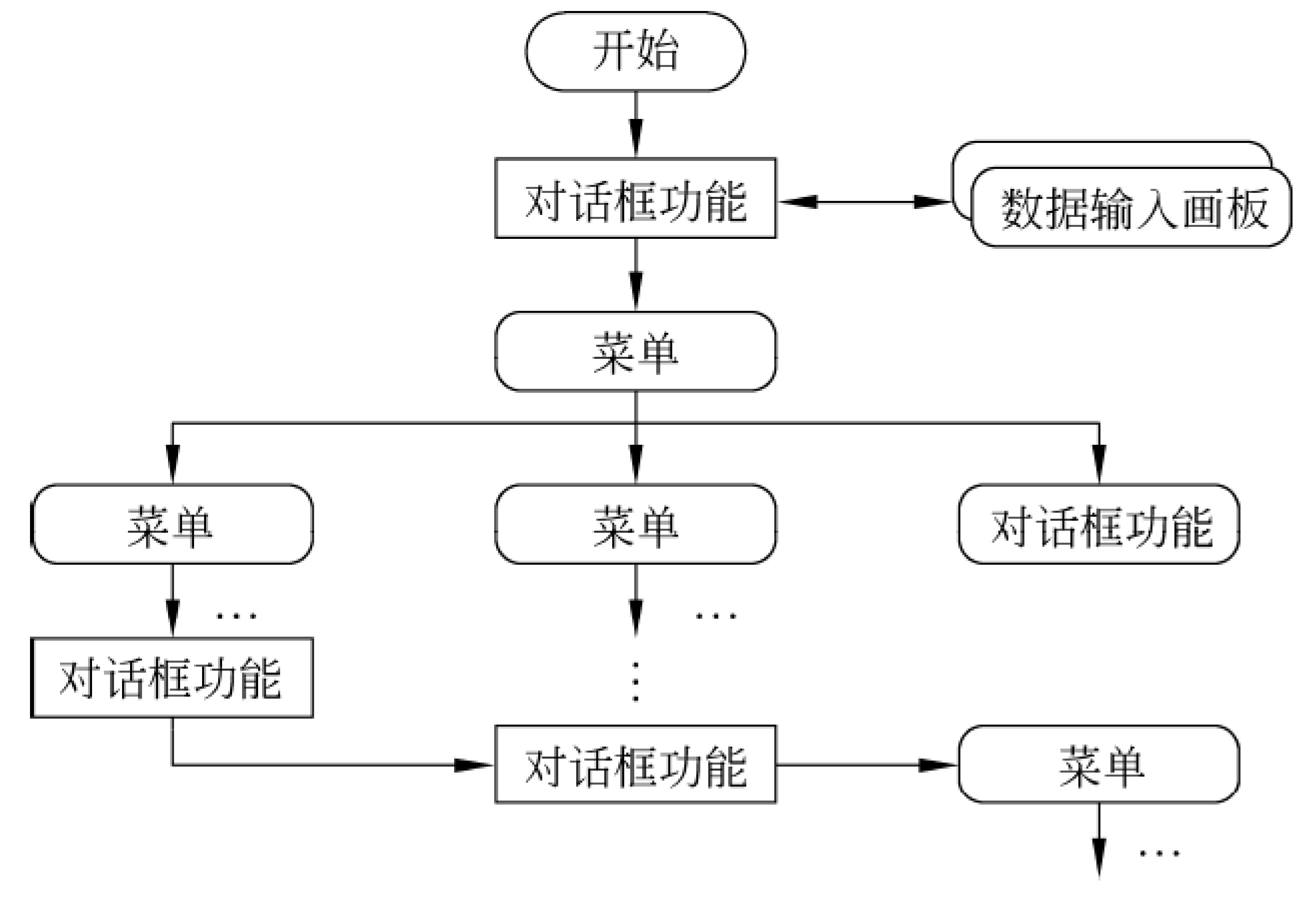


图 7-11　对话框组织方式 2②

7.4.3　什么是 SELECT 服务

ISPF 会话的功能指示程序中，通常需要调用 SELECT 服务，SELECT 服务是一个 ISPF

① 摘自 IBM 白皮书 *z/OS V1R6.0 ISPF Dialog Developer's Guide and Reference* 中图 6。
② 摘自 IBM 白皮书 *z/OS V1R6.0 ISPF Dialog Developer's Guide and Reference* 中图 7。

服务，它用来运行一个会话执行过程。用户通过指定 SELECT 服务的具体选项参数来确定会话执行的逻辑。通过在 REXX 程序中使用 SELECT 服务，可以通过执行程序或者命令来完成相应功能，或者通过调用面板来显示应用逻辑。

和 ISPSTART 命令一样，SELECT 命令同样也有 PANEL、CMD 和 PGM 三个参数选项。通过这三个选项，来指定会话程序在运行过程中调用的面板（panel）或者执行的功能（CMD 或者 PGM）。在 REXX 程序中使用 SELECT 命令来调用名为 ISPAPTT 的 ISPF 会话程序，例如：

```
'ISPEXEC SELECT PGM(ISPAPTT) PARM(APLTT)'
```

通常，在 REXX 程序中可以使用程序逻辑控制语句来完成 ISPF 会话的执行，流程如图 7-12 所示。

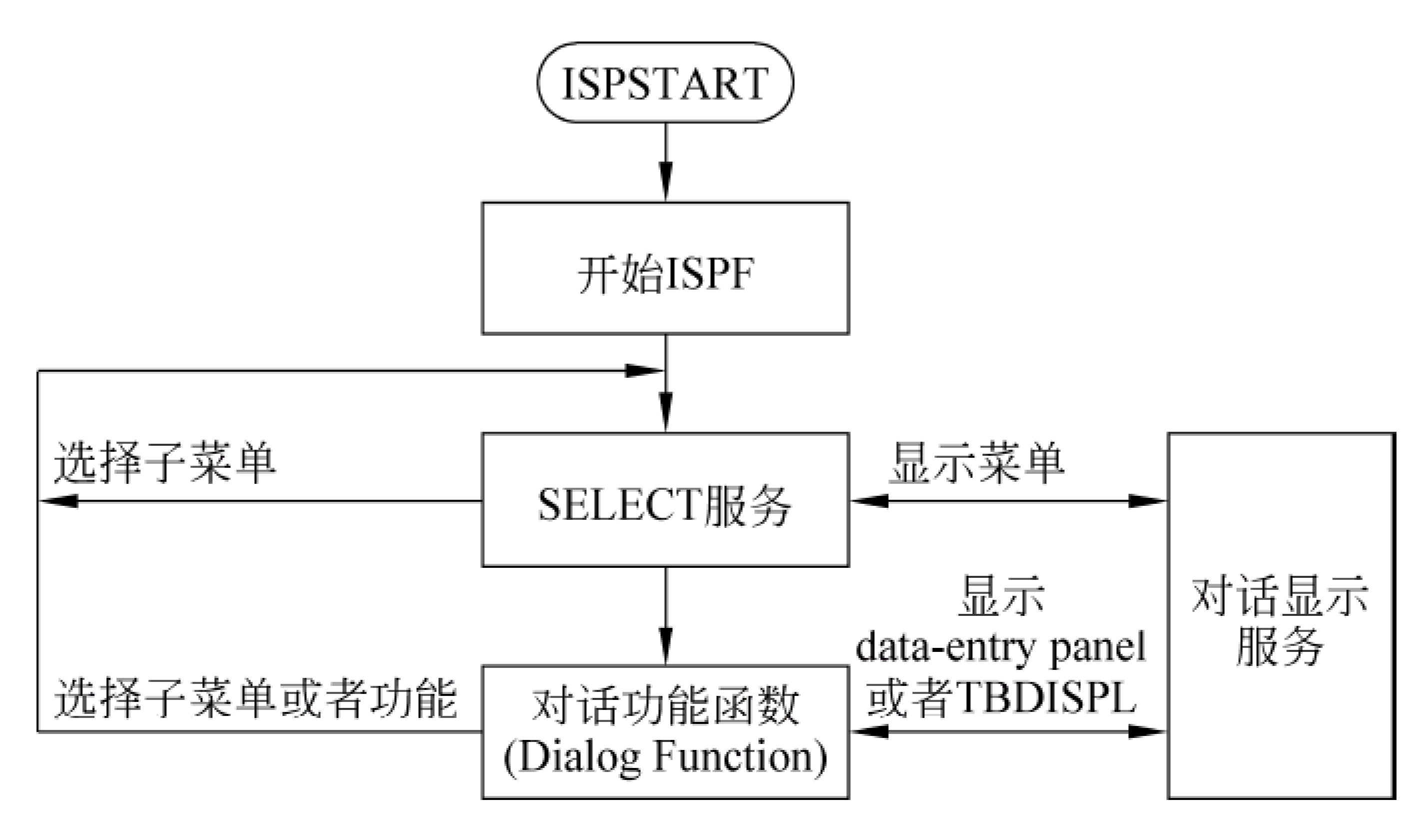


图 7-12　REXX 程序控制 ISPF 会话的执行

7.4.4　如何调用 SELECT 服务

SELECT 服务可以用以下几种方式调用：

（1）在初始化过程中，对话管理器会自动调用 SELECT 服务来启动第一个对话框。ISPSTART 命令最初指定的关键字会传递给 SELECT 服务。

说明：ISPSTART 调用的对话框时，直到 ISPF 已完全初始化，ISPF 的错误处理才生效。所以当确认执行后一直没有发生严重的错误处理，ISPF 才被认为是完全初始化了。

（2）如果进入分屏模式，对话框管理器再次调用 SELECT 服务，并再次从 ISPSTART 命令传递已选择的关键字给 SELECT 服务。这将导致已在 ISPSTART 命令中指定了参数的第一个屏幕，又在一个新的逻辑屏幕上被初始化。

（3）SELECT 服务可用递归的方式调用自身。当用户选择一个选项菜单 SELECT 服务调用了 SELECT 服务。在这种情况下，选择关键字在菜单的面板定义则被重新指定给 SELECT 服务。

（4）SELECT 服务可以从一个对话框函数中被调用。在这种情况下，选择关键字是通过调用顺序参数赋给 SELECT 服务的。

7.4.5　ISPSTART 命令启动 ISPF 对话

可以使用 ISPF、PDF 或 ISPSTART 命令，同时设置 CMD、PGM 或 PANEL 等关键字来启动 ISPF 或其他对话框。ISPF 是 TSO 环境下运行的命令过程。它可由 TSO 终端或 CLIST\REXX 命令过程来运行。在运行一个对话框之前，该对话框中所用到的数据集必须事先定义到 ISPF 库中。

1. ISPSTART 命令语法

通过 ISPSTART 命令可以来调用 ISPF 对话程序。用户通过 ISPSTART 命令的参数来指定首次显示的菜单或首次接收的函数。

假如参数没有被指定，ISPSTART 命令默认显示 DEFAULT_PRIMARY_PANEL 关键字指定的主屏幕，DEFAULT_PRIMARY_PANEL 关键字是在 ISPF 配置表中进行设置的。这个关键字通常设置为 ISP@MSTR。

通过如下命令可调用 ISPF 对话程序。

```
ISPSTART
{PANEL(panel-name) [OPT(option)][ADDPOP]}
{CMD(command parm1 parm2) [LANG(APL|CREX)]}
{PGM(program-name) [PARM(parameters)]}
{WSCMD(workstation-command)
[MODAL|MODELESS]
[WSDIR(dir)]
[MAX|MIN]
[VIS|INVIS]}
{WSCMDV(var_name)
[MODAL|MODELESS]
[WSDIR(dir)]
[MAX|MIN]
[VIS|INVIS] }
[GUI(LU:address:tpname|IP:address:port|,FI:)|,NOGUIDSP)]
[TITLE(title)]
[GUISCRW(screen-width)]
[GUISCRD(screen-depth)]
[FRAME(STD|FIX|DLG)]
[CODEPAGE(codepage)] [CHARSET(character_set)]
[BKGRND(STD|DLG)]
[NEWAPPL[(application-id)]]
[SCRNAME(screen-name)]
[TEST|TESTX|TRACE|TRACEX]
[NOLOGO|LOGO(logo-panel-name)]
[BATSCRW(screen-width)]
[BATSCRD(screen-depth)]
[BDISPMAX(max-number-of-displays)]
```

```
[BREDIMAX(max-number-of-redisplays)]
[BDBCS]
[DANISH|ENGLISH|GERMAN|JAPANESE|PORTUGUE|SPANISH|KOREAN|
FRENCH|ITALIAN|CHINESET|CHINESES|SGERMAN|UPPERENG]
```

示例：若调用指示功能名为 DLGPROC1 的对话，可通过输入命令：

```
ISPSTART CMD(DLGPROC1 parm1 parm2 …)
```

2. ISPSTART 命令使用

ISPSTART 命令可以指定要显示的第一个菜单，或者指定要接收控制的第一个功能程序。

示例：进入 ISPF 时首先调用名为 ABC 的面板，面板定义必须事先存储在面板库里。

```
ISPSTART PANEL(ABC)
```

示例：下例中调用 ISPF，指定对话框的处理是从一个名为 GHI 的程序函数开始。

```
ISPSTART PGM(GHI)
```

如果指定的 CMD（命令）或 ISPSTART 命令行关键字不止一次，那么 ISPF 使用指定的最后一个值。例如：

```
ISPSTART PANEL(PANELA) PANEL(PANELX)
```

ISPF 把上面的命令解释为：

```
ISPSTART PANEL(PANELX)
```

ISPSTART 命令通常是在 TSO 登录期间或在命令过程中书写。假设用户从 PCOMM 终端上登录 TSO，登录时调用一个名为 DBAPROC 的命令过程，过程 DBAPROC 首先为 ISPF 分配库，然后它使用 ISPSTART 命令开始 ISPF 的处理。过程 DBAPROC 则不能使用 ISPF 对话框服务，因为它尚未在 ISPF 下运行。

7.4.6 ISPF 对话框终止

当发生以下情况时，ISPF 对话框会终止。

（1）假如一个对话函数调用了 SELECT 服务，控制反馈给了函数并且函数能够继续执行。

（2）假如一个菜单调用了 SELECT 服务，当菜单被重复显示时，或者在面板定义中执行了 INIT 部分。

（3）如果终止了多屏模式，逻辑屏幕上原有的对话框结束，其他逻辑屏幕扩展到全屏。

（4）当终止 ISPF 时，显示 ISPF 终止面板或使用 ISPF SETTINGS 作为列表/日志处理的默认窗口。

如果有以下情况，ISPF 则会显示终止屏幕。

（5）对话框是由菜单的 DISPLAY 启动的，并且结束时在菜单上输入 END 命令。

（6）对话框是由执行函数启动的，并且函数结束的返回值为 0。

在终止对话框的时候，ISPF 返回给应用程序的错误码如下。

- 908：表示 ZISPFRC 值不可用。
- 920：表示 ISPSTART 命令语法不正确。
- 985：表示试图在批处理模式下启动一个 GUI，但是没有与 WORKSTATION 的连接。
- 987：表示试图用 GUISCRW 或者 GUISCRD 启动一个 GUI，但是 GUI 的初始化失败。
- 988：表示在初始化 IKJSATTN 过程中发生错误。
- 989：表示当 ISPF 仍然运行在 GUI 模式时，ISPF C/S 组件窗口被关闭。
- 990：表示在批处理模式下发生错误。假如 ZISPFRC 没有被预先设定，ISPF 发生一个严重的错误并终止了组件的运行。
- 997：表示不可纠正的 TPUT 错误。
- 998：表示 ISPF 初始化错误。产生 998 错误的原因可能为：
  - 要用到的 ISPF 数据元素库没有被预先准备好。
  - 没有正确地打开 ISPF 数据元素库。
  - ISPF 数据元素库包含不正确的数据集类型。
  - 错误导入文本模块。
  - 循环调用 ISPF 的 call 语句。
- 999：表示 ISPF 环境不可用。产生 999 错误的原因可能为：
  - TSO/MVS 不可用。
  - 不支持的屏幕尺寸。

7.5　ISPF 会话案例

7.5.1　ISPF 对话程序案例

业务逻辑：显示用户面板 TP01（见图 7-14），要求用户输入用户名，用户在面板 TP01 上输入用户名，然后回车，系统将显示面板 TP02（见图 7-16）欢迎用户并在面板 TP02 上提供 2 个选项让用户选择，选项 1 为提交 JCL 作业，选项 2 为进行 DB2 表操作。用户选择 1 系统则显示一个 JCL 作业内容，用户输入 SUBMIT 提交作业，系统则显示面板 TP03（见图 7-19）反馈作业提交结果；若用户选择 2 系统则显示面板 TP04（见图 7-23），提示用户输入 ID 回车后可以显示用户信息；如果用户选择其他选项则弹出一个错误信息框（见图 7-21）。

要求用户编写 5 个 REXX 程序。

- TR01：接受用户输入用户名跳往 TR02 程序。
- TR02：让用户选择执行并检查 JCL 或是操作 DB2。
- TR03：通过 REXX 直接操作 SDSF，找出指定 JOBID 的所有信息。
- TR04：通过 REXX 直接对 DB2 执行查询等操作。
- ERRPOPUP：弹出错误信息框。

通过该案例，用户可以学习：

- 基本的面板（Panel）画法。
- 如何在 REXX 中使用 TSO 命令和 ISPF 服务（通过 ISPEXEC 命令）。
- REXX 参数的传递。
- 使用 POP-UP 信息框。
- 熟悉 REXX 中逻辑的判断。

本案例中设计了 4 个 ISPF 面板 TP01、TP02、TP03 和 ERRORPOP 和 5 个 REXX 控制程序 TR01、TR02、TR03、TR04 和 ERRPOPUP。详细介绍如下。

1. 面板设计

假设面板定义存放在 TJA0095.REXX.PANEL 库（见图 7-13）中。

```
  Menu  Functions  Confirm  Utilities  Help
--------------------------------------------------------------------------------
ISRUDSM EDIT       TJA0095.REXX.PANEL                         Row 00001 of 00004
           Name      Prompt       Size    Created             Changed          ID
_________ ERRORPOP
_________ TP01
_________ TP02
_________ TP03
          **End**
```

图 7-13　TJA0095.REXX.PANEL 库

面板 1 TP01 运行效果和代码如图 7-14 所示。

```
TP01
                               TONGJI REXX SHOW

COMMAND ===>


                       Please enter your name to login




                               ID   TE02
                               NAME ________
```

图 7-14　TP01 面板示意

TP01 对应的面板定义代码如图 7-15 所示，用户在面板上输入的姓名将会放在变量

uname 中。

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
ISREDDE2   TJA0095.REXX.PANEL(TP01) - 01.00               Columns 00001 00072
****** ***************************** Top of Data ******************************
000001 )ATTR
000002   % TYPE(TEXT) COLOR(WHITE) CAPS(OFF) SKIP(ON)
000003   * TYPE(TEXT) CAPS(OFF) INTENS(HIGH) COLOR(GREEN) SKIP(ON)
000004   ! TYPE(TEXT) CAPS(OFF) INTENS(HIGH) COLOR(BLUE) SKIP(ON)
000005   _ TYPE(INPUT) PAD('_') INTENS(HIGH) COLOR(TURQ)
000006   @ TYPE(NEF) CAPS(ON) PADC(USER)
000007   # TYPE(OUTPUT) PAD(' ') CAPS(OFF) COLOR(YELLOW) JUST(LEFT)
000008   ? TYPE(NT)
000009 )BODY WIDTH(80) CMD(ZCMD) EXPAND(//)
000010 *
000011 !/ /TONGJI REXX SHOW/ /
000012 *
000013 *COMMAND ===>@ZCMD
000014 *
000015 *
000016 %/ /Please enter your name to login/ /
000017 *
000018 *
000019 *
000020 *
000021 */ /ID  #uid     * / /
000022 */ /NAME_uname   * / /
000023 *
000024 )PROC
000025  VER(&uname,NB)
000026 )end
****** **************************** Bottom of Data ****************************
```

图 7-15　TP01 面板定义代码

面板 2 TP02 运行效果和代码如图 7-16 所示。

```
TP02
                                JOB SELECT PANEL

COMMAND ===>


                   Welcom, GAOZHEN ,Please select what to do



                     Your Choice: _
                     1.Submit a JCL and check its output
                     Please submit the job and record JOBID

                     2.Select data from a DB2 table
```

图 7-16　TP02 面板示意

TP02 对应的面板定义代码如图 7-17 所示。选项内容将放在变量 C 中。

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
ISREDDE2   TJA0095.REXX.PANEL(TP02) - 01.00                Columns 00001 00072
****** ***************************** Top of Data ******************************
)ATTR
  % TYPE(TEXT) COLOR(WHITE) CAPS(OFF) SKIP(ON)
  * TYPE(TEXT) CAPS(OFF) INTENS(HIGH) COLOR(GREEN) SKIP(ON)
  ! TYPE(TEXT) CAPS(OFF) INTENS(HIGH) COLOR(BLUE) SKIP(ON)
  _ TYPE(INPUT) PAD('_') INTENS(HIGH) COLOR(TURQ)
  @ TYPE(NEF) CAPS(ON) PADC(USER)
  # TYPE(OUTPUT) PAD(' ') CAPS(OFF) COLOR(YELLOW) JUST(LEFT)
  ? TYPE(NT)
)BODY WIDTH(80) CMD(ZCMD) EXPAND(//)
*
!/ /JOB SELECT PANEL/ /
*
*COMMAND ===>@ZCMD
*
*
*/ /Welcom,#uname  *,Please select what to do/ /
*
*
*
*/ / Your Choice:_C*                        / /
*/ /%1.Submit a JCL and check its output    / /
!/ / Please submit the job and record JOBID/ /
*
*/ /%2.Select data from a DB2 table         / /
*
*
*
)PROC
 VER(&C,NB)
)end
****** **************************** Bottom of Data ****************************
```

图 7-17　TP02 面板定义代码

如果在 TP02 面板中选择 1 提交 JCL 作业，则系统会显示如图 7-18 所示的 JCL 作业，用户可以在命令行上输入 SUBMIT 提交作业。

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
ISREDDE2   TJA0095.REXX.JCL(VSAMDEF) - 01.01                Columns 00001 00072
****** ***************************** Top of Data ******************************
//TJA0095A JOB ACCT#,'SHICHU',CLASS=A,NOTIFY=&SYSUID,
//          MSGCLASS=A,MSGLEVEL=(1,1)
//STEP1 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  DELETE TJA0095.VSAM.KSDS1
  SET MAXCC=0
  DEFINE CLUSTER -
        (NAME(TJA0095.VSAM.KSDS1) -
        INDEXED -
```

图 7-18　JCL 作业代码

```
001100         VOLUME(BX3D61) -
001200         RECORDS(100,10) -
001300         RECORDSIZE(10 20) -
001400         CONTROLINTERVALSIZE(4096) -
001500         KEYS(1,1))
001600 //
****** **************************** Bottom of Data ***************************


Command ===> SUBMIT                                              Scroll ===> CSR
```

图 7-18（续）

作业提交之后系统会显示面板 TP03，显示作业的提交结果，如图 7-19 所示。

```
TP03
                                 HELLO

COMMAND ===>


                              Check the jobs output



                      JobID: J0012479

                      JOBNAME : TE02A
                      JOBOWNER: TE02
                      RENTCODE: CC 0000

                      Information:
```

图 7-19　TP03 面板示意

TP03 面板的定义代码如图 7-20 所示。

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
ISREDDE2   TJA0095.REXX.PANEL(TP03) - 01.00                  Columns 00001 00072
****** ***************************** Top of Data ******************************
000001 )ATTR
000002   % TYPE(TEXT) COLOR(WHITE) CAPS(OFF) SKIP(ON)
000003   * TYPE(TEXT) CAPS(OFF) INTENS(HIGH) COLOR(GREEN) SKIP(ON)
000004   ! TYPE(TEXT) CAPS(OFF) INTENS(HIGH) COLOR(BLUE) SKIP(ON)
000005   _ TYPE(INPUT) PAD('_') INTENS(HIGH) COLOR(TURQ)
000006   @ TYPE(NEF) CAPS(ON) PADC(USER)
000007   # TYPE(OUTPUT) PAD(' ') CAPS(OFF) COLOR(YELLOW) JUST(LEFT)
000008   ? TYPE(NT)
000009 )BODY WIDTH(80) CMD(ZCMD) EXPAND(//)
000010 *
000011 !/ /HELLO #USER/ /
000012 *
000013 *COMMAND ===>@ZCMD
```

图 7-20　TP03 面板定义代码

```
*
*
%/ /Check the jobs output/ /
*
*
*
*/ /JobID:_jobidt  *                    / /
*
*/ /JOBNAME :#jobname *                 / /
*/ /JOBOWNER:#jobowner*                 / /
*/ /RENTCODE:#retncode*                 / /
*
*/ /Information:#inform                 / /
*
*
)end
****** **************************** Bottom of Data ****************************
```

图 7-20（续）

如果在 TP02 面板中选择其他选项，如 3，则系统会弹出错误信息框，如图 7-21 所示。

```
|---------------------------------------------|
| ERRPOPUP  Please enter again!               |
|---------------------------------------------|
```

图 7-21 错误信息框示意

图 7-21 对应的信息框 ERRPOPUP 定义代码如图 7-22 所示。

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
ISREDDE2    TJA0095.REXX.PANEL(ERRPOPUP) - 01.00          Columns 00001 00072
****** ***************************** Top of Data ******************************
)ATTR
  % TYPE(TEXT) COLOR(WHITE) CAPS(OFF) SKIP(ON)
  * TYPE(TEXT) CAPS(OFF) INTENS(HIGH) COLOR(GREEN) SKIP(ON)
  ! TYPE(OUTPUT) PAD(' ') CAPS(OFF) COLOR(YELLOW) JUST(LEFT)
  _ TYPE(INPUT) PAD(' ') INTENS(HIGH) COLOR(TURQ)
  @ TYPE(NEF) CAPS(ON) PADC(USER)
  # TYPE(AB)
  ? TYPE(NT)
)BODY WINDOW(40,1)
!POPERR
)END
****** **************************** Bottom of Data ****************************
```

图 7-22 错误信息框定义代码

如果在 TP02 面板中选择 2 在 DB2 表里插入数据，则系统会显示面板 TP4，如图 7-23 所示。查询 DB2 表数据，根据用户 ID 显示用户姓名等信息。

TP04 面板的定义代码如图 7-24 所示。

```
TP04                                   SDSF JOB STATUS

 COMMAND ===>


                                 CHECK USER INFORMATION


                         PLEASE INPUT USER ID HERE:
                         USERID: ____

                         USER NAME   :
                         USER MOBILE:

                         SQLCODE     :
```

图 7-23　数据库操作面板

```
   File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
 EDIT       TJA0095.REXX.PANEL(TP04) - 01.01                Columns 00001 00072
 ****** **************************** Top of Data ******************************
 )ATTR
   % TYPE(TEXT) COLOR(WHITE) CAPS(OFF) SKIP(ON)
   * TYPE(TEXT) CAPS(OFF) INTENS(HIGH) COLOR(GREEN) SKIP(ON)
   ! TYPE(TEXT) CAPS(OFF) INTENS(HIGH) COLOR(BLUE) SKIP(ON)
   _ TYPE(INPUT) PAD('_') INTENS(HIGH) COLOR(TURQ)
   @ TYPE(NEF) CAPS(ON) PADC(USER)
   # TYPE(OUTPUT) PAD(' ') CAPS(OFF) COLOR(YELLOW) JUST(LEFT)
   ? TYPE(NT)
 )BODY WIDTH(80) CMD(ZCMD) EXPAND(//)
 *
 !/ /SDSF JOB STATUS/ /
 *
 *COMMAND ===>@ZCMD
 *
 *
 %/ /CHECK USER INFORMATION/ /
 *
 *
 */ /PLEASE INPUT USER ID HERE:         / /
 */ /USERID:_RID *                      / /
 *
 */ /USER NAME   :#RNAME               */ /
 */ /USER MOBILE:#RMOBILE              */ /

 */ /SQLCODE     :#INFORM               / /
 *
 *
 )END
 ****** *************************** Bottom of Data ****************************
```

图 7-24　TP04 面板定义代码

2. REXX 控制程序

实现业务逻辑的 REXX 控制程序存放在 TJA0095.REXX 库中，该库中有 5 个成员，如图 7-25 所示。其中成员 TR01 用于接受用户输入用户名跳往 TR02 程序；成员 TR02 用于让用户选择执行并检查 JCL 或是操作 DB2；成员 TR03 用于通过 REXX 直接操作 SDSF，找出指定 JOBID 的所有信息；成员 TR04 通过 REXX 直接对 DB2 执行查询等操作；成员 ERRPOPUP 用于弹出错误信息框。

```
  Menu  Functions  Confirm  Utilities  Help
--------------------------------------------------------------------------------
ISRUDSM EDIT       TJA0095.REXX                              Row 00001 of 00008
           Name     Prompt        Size   Created           Changed          ID
_________ ERRPOPUP
_________ TR01                      21  2011/10/10  2011/10/12 18:31:01  TJA0095
_________ TR02
_________ TR03                      33  2011/10/10  2011/10/17 00:45:24  TE02
_________ TR04
          **End**
```

图 7-25　TJA0095.REXX 库

TR01 程序内容如图 7-26 所示。

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
ISREDDE2   TJA0095.REXX(TR01) - 01.02                          Columns 00001 00072
****** ***************************** Top of Data *****************************
000001 /* REXX */
000003 uid=USERID()
000005 ADDRESS TSO
000006 "Profile noprefix"
000008 ADDRESS ISPEXEC
000009 "CONTROL ERRORS RETURN"
000011 /* Define the panel lib, from where panels are retrieved */
000012 panellib = 'tja0095.REXX.PANEL'
000013 "ISPEXEC LIBDEF ISPPLIB DATASET ID('"panellib"')"
000015 /* Send panel TP01 from the ISPPLIB */
000016 "DISPLAY PANEL(TP01)"
000018 /* Afert you've inserted your name, call the main choice panel */
000019 RC=TR02(uname)
000021 RETURN 0
****** **************************** Bottom of Data ***************************
```

图 7-26　TR01 程序内容

TR02 程序内容如图 7-27 所示。

TR03 程序内容如图 7-28 所示。

TR04 程序内容如图 7-29 所示。

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
ISREDDE2   TJA0095.REXX(TR02) - 01.00                      Columns 00001 00072
****** ***************************** Top of Data ******************************
/* REXX */
uname=Arg(1)
ADDRESS ISPEXEC
"DISPLAY PANEL(TP02)"
RC=4
/* If the choice is neither 1 or 2, send a pop up window */
DO WHILE RC<>0
  SELECT
    WHEN C=1 THEN DO
      RC=TR03()
    END
    WHEN C=2 THEN DO
      RC=TR04()
    END
    OTHERWISE DO
      ERRPOPUP()
      C=''
      "DISPLAY PANEL(TP02)"
      RC=4
    END
  END
END
RETURN RC
****** **************************** Bottom of Data ****************************
```

图 7-27　TR02 程序内容

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
ISREDDE2   TJA0095.REXX(TR03) - 01.02                      Columns 00001 00072
****** ***************************** Top of Data ******************************
/* REXX */
/*View a dataset then submit it*/
Address ISPEXEC
  "view dataset("'tja0095.REXX.JCL(VSAMDEF)'")"
/*Display the panel for the first time*/
ADDRESS ISPEXEC
"DISPLAY PANEL(TP03)"
/*Access SDSF via REXX*/
/*To get the job's information from the JOBID we specified*/
rc=isfcalls("ON")
isfowner=USERID()

ADDRESS SDSF "ISFEXEC ST"
If rc<>0 Then Exit 20

jobfind=0
Do ix=1 to isfrows
  If JOBID.ix=jobidt Then Do
     jobname=JNAME.ix
     jobowner=OWNERID.ix
     retncode=RETCODE.ix
     jobfind=1
  End
End
```

图 7-28　TR03 程序内容

```
rc=isfcalls("OFF")
/*Verify whether the job is found or not*/
If jobfind=0 Then Do
   inform=jobidt "not found!"
End
/*Display the panel for the second time*/
ADDRESS ISPEXEC
"DISPLAY PANEL(TP03)"
RETURN 0
****** **************************** Bottom of Data ****************************
```

图 7-28（续）

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
ISREDDE2   TJA0095.REXX(TR04) - 01.02                         Columns 00001 00072
****** ***************************** Top of Data ******************************
/* REXX */

TRACE O
RSCHEMA='TJA0095';
ADDRESS ISPEXEC
"DISPLAY PANEL(TP04)"

SQLSTMT = "SELECT NAME,MOBILE "||,
          "FROM "||RSCHEMA||"."||"TBINFO "||,
          "WHERE UID ='"||RID||"';"

ADDRESS TSO
'SUBCOM DSNREXX'
IF RC THEN DO
  S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX')
END
ELSE DO
  S_RC = RXSUBCOM('DELETE','DSNREXX','DSNREXX')
  S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX')
END

ADDRESS DSNREXX
'CONNECT' DP1A
"EXECSQL DECLARE C1 CURSOR FOR S1"
"EXECSQL PREPARE S1 INTO :OUTSQLDA FROM :SQLSTMT"

"EXECSQL OPEN C1"
   IF SQLCODE = 0 THEN DO
      "EXECSQL FETCH C1 USING DESCRIPTOR :OUTSQLDA"
      RNAME = OUTSQLDA.1.SQLDATA
      RMOBILE = OUTSQLDA.2.SQLDATA
      INFORM = SQLCODE
   END
   ELSE DO
     INFORM=SQLCODE
   END

"EXECSQL CLOSE C1"

ADDRESS ISPEXEC
"DISPLAY PANEL(TP04)"

RETURN 0
```

图 7-29　TR04 程序内容

ERRPOPUP 程序内容如图 7-30 所示。

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
ISREDDE2   TJA0095.REXX(ERRPOPUP) - 01.00                    Columns 00001 00072
****** ***************************** Top of Data ******************************
000100 /*REXX*/
000200
000600 /*ERRPOP, pops up an error message*/
000900 Errmsg = Arg(1)
001000 Address ISPEXEC
001100 If (Errmsg = '') Then Do
001200   Errmsg = 'Wrong input! Please enter again!'
001300 End
001400 POPERR = Errmsg
001500 "ISPEXEC ADDPOP"
001600 ZCMD = ''
001700 "ISPEXEC DISPLAY PANEL(ERRPOPUP)"
001800 "ISPEXEC REMPOP"
001900 return 0
****** **************************** Bottom of Data ****************************
```

图 7-30　ERRPOPUP 程序内容

3. DB2 数据表设计

为了成功执行上面的 REXX 程序进行 DB2 数据表的操作，必须先在 DB2 里面创建出相应的数据表。参考代码如下。

```
SET CURRENT SCHEMA = 'TJA0095';
CREATE DATABASE DBSP29;
CREATE TABLESPACE TSINFO
  IN DBSP29
  LOCKSIZE ROW
  LOCKMAX  0
  SEGSIZE  4
  FREEPAGE 5
  PCTFREE 5
  CLOSE NO;
CREATE TABLE TBINFO(
  UID    CHAR(4),
  NAME   CHAR(20),
  MOBILE CHAR(11)
  ) IN DBSP29.TSINFO;
COMMIT;
INSERT INTO TBINFO VALUES('1','SHICHU','136XXXXXXXX');
INSERT INTO TBINFO VALUES('2','KAKA','136XXXXXXXX');
INSERT INTO TBINFO VALUES('3','PATO','136XXXXXXXX');
INSERT INTO TBINFO VALUES('4','PIRLO','136XXXXXXXX');
INSERT INTO TBINFO VALUES('5','MESSI','136XXXXXXXX');
SELECT * FROM TBINFO;
```

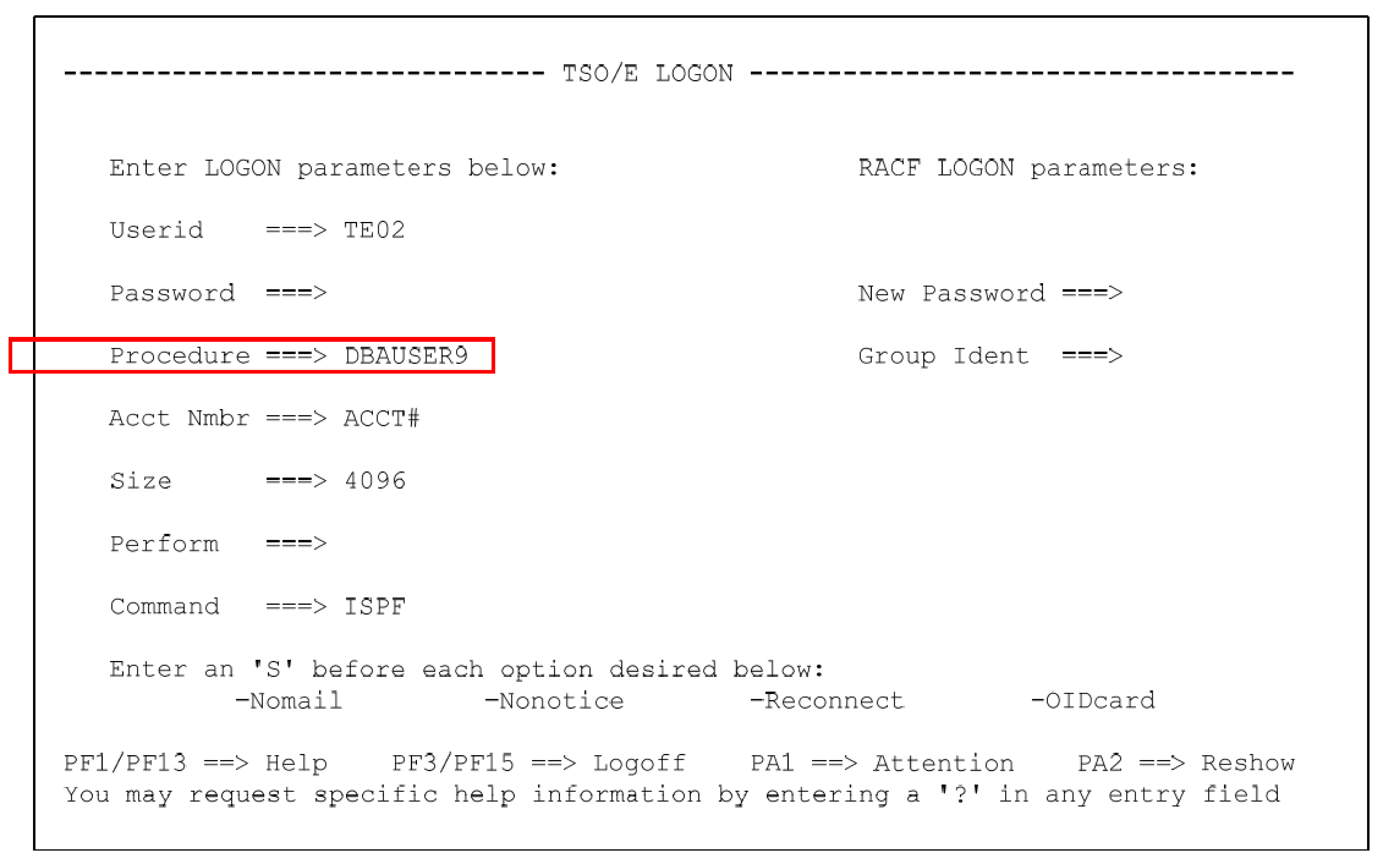
7.5.2 客户化 ISPF 主面板

用户有时需要将 ISPF 主面板客户化或者把已经编写好的 ISPF 对话程序嵌入到 ISPF 主面板上，尤其是后者，是在完成 ISPF 对话程序之后常常遇到的难题。下面进行介绍。

为了修改 ISPF 主面板，首先要定位 ISPF 主面板的位置。每个 TSO 登录过程关联的 ISPF 主面板可能不同，如果要定位 ISPF 主面板，首先要找到 TSO 登录过程。

定位 TSO 登录过程的步骤如下：

① 在 TSO 登录面板上找到 TSO 登录过程的名字。某系统示例如图 7-31 所示。



```
------------------------------ TSO/E LOGON -----------------------------------

   Enter LOGON parameters below:                       RACF LOGON parameters:

   Userid    ===> TE02

   Password  ===>                                      New Password ===>

   Procedure ===> DBAUSER9                             Group Ident  ===>

   Acct Nmbr ===> ACCT#

   Size      ===> 4096

   Perform   ===>

   Command   ===> ISPF

   Enter an 'S' before each option desired below:
           -Nomail         -Nonotice        -Reconnect        -OIDcard

PF1/PF13 ==> Help    PF3/PF15 ==> Logoff    PA1 ==> Attention    PA2 ==> Reshow
You may request specific help information by entering a '?' in any entry field
```

图 7-31　TSO 登录面板

② 在 JES 的过程库中寻找 DBAUSER9 成员。若想寻找 JES 的过程库，首先要找到 JES 自身的启动过程，一般 JES 的启动过程会放在 SYS1.PROCLIB 中。下面看看如何准确定位 JES 的启动过程。

定位 JES 的启动过程步骤如下：

① 使用 z/OS 命令 D IPLINFO 和 D PARMLIB 查找系统的启动参数和参数库。由图 7-32 可知，系统的参数文件为 IEASYS00。

由图 7-33 可知，系统的参数库为 USER.PARMLIB 和 SYS1.PARMLIB。

② 在系统过程库（PARMLIB）中查找 IEASYS00，某系统示例如图 7-34 所示。

```
  Display  Filter  View  Print  Options  Help
 -------------------------------------------------------------------------------
HQX7730 -----------------   SDSF PRIMARY OPTION MENU    -- COMMAND ISSUED
RESPONSE =TESTMVS
 IEE254I  03.00.48 IPLINFO DISPLAY  140
  SYSTEM IPLED AT  21.52.33 ON  09/07/2011
  RELEASE z/OS 01.08.00     LICENSE  = z/OS
  USED LOAD 31  IN SYS1.IPLPARM ON  0F9F
  ARCHLVL  = 2    MTLSHARE  = N
  IEASYM LIST  = DP
  IEASYS LIST  = 00  (OP)
  IODF DEVICE  0F9F
  IPL DEVICE  0C80  VOLUME DMTRES
JC     Job classes                         RM     Resource monitor
SE     Scheduling environments             CK     Health checker
RES    WLM resources
ENC    Enclaves                            ULOG   User session log
PS     Processes

END    Exit SDSF




COMMAND INPUT  ===> /D IPLINFO                                   SCROLL  ===> CSR
```

图 7-32　D IPLINFO 命令执行结果

```
  Display  Filter  View  Print  Options  Help
 -------------------------------------------------------------------------------
HQX7730 -----------------   SDSF PRIMARY OPTION MENU    -- COMMAND ISSUED
RESPONSE =TESTMVS
 IEE251I 03.02.38 PARMLIB DISPLAY  144
  PARMLIB DATA SETS SPECIFIED
  AT IPL
  ENTRY  FLAGS  VOLUME  DATA SET
    1        S     DMTP07   USER.PARMLIB
    2        S     DMTRES  SYS1.PARMLIB
LOG    System log                          SO     Spool offload
SR     System requests                     SP     Spool volumes
MAS    Members in the MAS
JC     Job classes                         RM     Resource monitor
SE     Scheduling environments             CK     Health checker
RES    WLM resources
ENC    Enclaves                            ULOG   User session log
PS     Processes

END    Exit SDSF


COMMAND INPUT  ===> /D PARMLIB                                     SCROLL  ===> CSR
```

图 7-33　D PARMLIB 命令执行结果

由图 7-34 可知，系统的第一个地址空间的启动过程是 MSTJCL00。

③ 在系统参数库中寻找 MSTJCL00 成员，某系统示例如图 7-35 所示。

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
VIEW       SYS1.PARMLIB(IEASYS00) - 01.99                    CHARS 'JCL' found
****** ***************************** Top of Data ******************************
000001 CLOCK=00,
000002 CON=00,                       SELECT CONSOL00
000003 GRSCNF=00,                    SELECT GRSCNF00
000004 GRSRNL=00,                    SELECT GRSRNL00
000005 GRS=NONE,                     GRS CONFIG
000006 PLEXCFG=MONOPLEX,             SYSPLEX CONFIG
000007 ALLOC=00,                     SELECT ALLOC00
000008 CMD=UR,                       SELECT COMMND00
000009 COUPLE=00,                    SELECT COUPLE00
000010 DEVSUP=00,                    SELECT DEVSUP00
000011 FIX=00,                       SELECT IEAFIX00
000012 ICS=00,                       SELECT IEAICS00
000013 IOS=00,                       SELECT IECIOS00
000014 LNK=00,                       SELECT LNKLST00
000015 LPA=00,                       SELECT LPALST00
000016 MLPA=00,                      SELECT IEALPA00
000017 MSTRJCL=00,                   SELECT MSTRJCL00
000018 OMVS=(00,FS),                 SELECT BPXPRM00
000019 OPT=00,                       SELECT IEAOPT00


Command ===>                                                      Scroll ===> CSR
```

图 7-34　IEASYS00 参数内容

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
VIEW       SYS1.PARMLIB(MSTJCL00) - 01.03                  Columns 00001 00072
****** ***************************** Top of Data ******************************
000100 //MSTJCL00 JOB MSGLEVEL=(1,1),TIME=1440
000200 //         EXEC PGM=IEEMB860,DPRTY=(15,15)
000300 //STCINRDR DD SYSOUT=(A,INTRDR)
000400 //TSOINRDR DD SYSOUT=(A,INTRDR)
000410 //IEFPARM  DD DSN=SYS1.PARMLIB,DISP=SHR
000420 //IEFPDSI  DD DSN=SYS1.PROCLIB,DISP=SHR
000421 //         DD DSN=CENTER.PROCLIB,DISP=SHR
000430 //IEFJOBS  DD DSN=CENTER.JOBLIB,DISP=SHR
000440 //SYSUADS  DD DSN=SYS1.UADS,DISP=SHR
000450 //SYSLBC   DD DSN=SYS1.BRODCAST,DISP=SHR
****** **************************** Bottom of Data ****************************


Command ===>                                                      Scroll ===> CSR
```

图 7-35　MSTJCL00 成员内容

由图 7-35 可知，JES 的启动过程位于 IEFPDSI 对应的 SYS1.PROCLIB 或 CENTER.PROCLIB 库中。

④ 在 IEFPDSI 库中查找 JES2 成员，某系统示例如图 7-36 所示。

由图 7-36 可知，JES2 的过程库有 7 个，从 USER.PROCLIB 到 SYS1.LOGON。

TSO 登录过程就放在 JES2 的过程库中。某系统示例如图 7-37 所示。

由图 7-37 可知，TSO 登录过程会调用一个 CICS/REXX 程序 DBAUSER9，DBAUSER9

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
VIEW       SYS1.PROCLIB(JES2) - 01.17                    Columns 00001 00072
****** ***************************** Top of Data ******************************
000100 //JES2       PROC P=JES2SP,N=JES2NODE
000200 //*********************************************************************
000300 //* JES2 PROC
000400 //*********************************************************************
000410 //JES2       EXEC PGM=HASJES20,TIME=1440
000420 //*ES2       EXEC PGM=HASJES20,TIME=1440,PARM=(WARM,NOREQ)
000600 //HASPLIST   DD DDNAME=IEFRDER
000700 //HASPPARM   DD DSN=SYS1.TESTPLX.PARMLIB(&P),DISP=SHR
000800 //           DD DSN=SYS1.TESTPLX.PARMLIB(&N),DISP=SHR
000900 //           DD DSN=SYS1.SHASPARM,DISP=SHR
001010 //PROC00     DD DSN=USER.PROCLIB,DISP=SHR
001011 //           DD DSN=CENTER.PROCLIB,DISP=SHR
001020 //           DD DSN=SYS1.PROCLIB,DISP=SHR
001200 //           DD DSN=TIVOLI.PROCLIB,DISP=SHR
001300 //           DD DSN=CANDLET.XEGA.PROCLIB,DISP=SHR
001320 //           DD DSN=TJCICS.PROCLIB,DISP=SHR
001400 //PROC01     DD DSN=SYS1.LOGON,DISP=SHR
001500 //IEFRDER    DD DUMMY
****** **************************** Bottom of Data ****************************

Command ===>                                                  Scroll ===> CSR
```

图 7-36 JES2 的启动过程

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
VIEW       SYS1.LOGON(DBAUSER9) - 01.01                  Columns 00001 00072
****** ***************************** Top of Data ******************************
000100 //DBAUSER  PROC
000200 //*********************************************************************
000300 //* DBAUSER LOGON PROCEDURE     (USED FOR DATAMGMT FUNCTIONS)
000400 //* ALL ALLOCATIONS ARE DONE IN "CENTER.CLIST(DBAUSER)"
000500 //*********************************************************************
000600 //DBAUSER  EXEC PGM=IKJEFT01,DYNAMNBR=500,PARM='%DBAUSER9'
000700 //STEPLIB  DD DISP=SHR,DSN=VIP.V61.VANLOAD
000800 //         DD DISP=SHR,DSN=CANDLET.AOIRM.TKAULOAD
000900 //         DD DISP=SHR,DSN=CANDLET.AOIRM.TKANMOD
001000 //         DD DISP=SHR,DSN=CANDLET.AOIRM.TKANMODL
001100 //         DD DISP=SHR,DSN=CANDLET.AOIRM.TKANMODS
001200 //SYSPROC  DD DSN=TE02.REXX.LAB,DISP=SHR
001210 //         DD DSN=CENTER.CLIST,DISP=SHR
001300 //SYSHELP  DD DSN=SYS1.HELP,DISP=SHR
001400 //SYSLBC   DD DSN=SYS1.BRODCAST,DISP=SHR
001500 //SYSPRINT DD TERM=TS,SYSOUT=X
001600 //SYSTERM  DD TERM=TS,SYSOUT=X
001700 //SYSIN    DD TERM=TS
001800 //*
****** **************************** Bottom of Data ****************************
```

图 7-37 TSO 登录过程内容

放在 ISPF 的 SYSPROC 或 SYSEXEC 库中，可以在 ISPF 的命令栏中使用 DDLIST 命令查看系统的 SYSPROC 或 SYSEXEC 库内容，某系统 DBAUSER9 示例如图 7-38 所示。

从图 7-38 可知，ISPF 的主面板是 ISR@390 成员，它会放在 ISPPLIB 库中。另外，还有一个简单的方法也可以获悉当前 ISPF 主面板的成员名称，就是在 ISPF 命令栏中输入 PANELID。示例如图 7-39 所示。

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
---------------------------------------------------------------------------------
VIEW       CENTER.CLIST(DBAUSER9) - 01.14                   Columns 00001 00072
****** ***************************** Top of Data ******************************
000100 PROC 0
000200 /****************************************************************/
000300 /* DBAUSER LOGON CLIST                                           */
000400 /* USED WITH DATA MANAGEMENT                                     */
000500 /****************************************************************/
000600 CONTROL MAIN NOMSG NOFLUSH NOLIST NOCONLIST
000700 PROFILE MODE WTPMSG MSGID
000800 FREE FILE(ISPLLIB,ISPPLIB,ISPMLIB,ISPTLIB,ISPSLIB, +
000900           SYSEXEC,ISPPROF,ISPTABL)
001000 /****************************************************************/
001100 /* ALLOCATE ISPPROF DATASET                                      */
001200 /****************************************************************/
001300 SET &DSNAME  = &SYSUID..ISPF.ISPPROF
.................................................................
029000 /****************************************************************/
029100 /* START ISPF                                                    */
029200 /****************************************************************/
029300 PDF PANEL(ISR@390)
029400 /****************************************************************/
029500 EXIT
****** **************************** Bottom of Data ****************************
```

图 7-38　TSO 登录过程调用的 CLIST 程序

```
  Menu  Utilities  Compilers  Options  Status  Help
------------------------------------------------------------------------------
ISR@390                      z/OS Primary Option Menu

0  Settings      Terminal and user parameters          User ID    . : TE02
1  View          Display source data or listings       Time   . . . : 03:40
2  Edit          Create or change source data          Terminal   . : 3278
3  Utilities     Perform utility functions             Screen   . . : 1
4  Foreground    Interactive language processing       Language   . : ENGLISH
5  Batch         Submit job for language processing    Appl ID    . : ISP
6  Command       Enter TSO or Workstation commands     TSO logon    : DBAUSER
7  Dialog Test   Perform dialog testing                TSO prefix   :
8  DB2            DB2 SPUFI                             System ID   : TESTMVS
9  IBM Products  IBM program products                  MVS acct   . : ACCT#
10 SCLM          SW Configuration Library Manager      Release    . : ISPF 5.8
11 Workplace     ISPF Object  /Action Workplace
12 z/OS System    z /OS system programmer applications
13 z/OS User      z /OS user applications
M  Minibank      Minibank Control Panel

     Enter X to Terminate using log  /list defaults


Option  ===> PANELID
```

图 7-39　查看面板名称

ISPF 的面板会存放在 ISPF 的 ISPPLIB 库中。使用 DDLIST 命令查看某系统的 ISPPLIB 库。示例如图 7-40 所示。

最后可以成功在某 ISPPLIB 中定位 ISPF 主面板。

假设想在 ISPF 主面板上增加一个新的入口（Entry），关联到一个 ISPF 对话程序，在 ISR@390 面板定义中增加两条语句（如图 7-41 所示），可以为 Minibank ISPF 对话程

```
                       Current Data Set Allocations          Row 72 of 204

  Volume    Disposition Act DDname    Data Set Name   Actions: B E V M F C I Q
  DMTCAT    SHR,KEEP     >  ISPPLIB   CENTER.ISPPLIB
  DMTP09    SHR,KEEP     >            SCLMSAW.V2R1.SAUZPENU
  DMTRES    SHR,KEEP     >            ISP.SISPPENU
  DMTRES    SHR,KEEP     >            SYS1.SBLSPNL0
  DMTRES    SHR,KEEP     >            SYS1.SBPXPENU
  DMTOS4    SHR,KEEP     >            BOOKMAN.SEOYPENU
  DMTP09    SHR,KEEP     >            DEBUG.V7R1.SEQAPENU
  DMTOS1    SHR,KEEP     >            CBC.SCBCPNL
  DMTRES    SHR,KEEP     >            SYS1.DGTPLIB
  DMTRES    SHR,KEEP     >            SYS1.DFQPLIB
  DMTRES    SHR,KEEP     >            SYS1.SEDGPENU
  DMTOS1    SHR,KEEP     >            DFSORT.SICEPENU
  DMTOS3    SHR,KEEP     >            FFST.SEPWPENU


 Command ===>  DDLIST                                      Scroll ===> PAGE
  F1=Help     F2=Split    F3=Exit     F5=Rfind    F7=Up       F8=Down     F9=Swap
 F10=Left    F11=Right   F12=Cancel
```

图 7-40 DDLIST 查看 ISPF 各类库的信息

```
   File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
 -------------------------------------------------------------------------------
 ISREDDE2   CENTER.ISPPLIB(ISR@390) - 01.31                 Columns 00001 00072
 ****** ***************************** Top of Data ******************************
 000001 )PANEL KEYLIST(ISRSAB,ISR) IMAGE(&ZIMGNAM,&ZIMGROW,&ZIMGCOL)
 000002 )ATTR DEFAULT(   ) FORMAT(MIX)               /* ISR@390 - ENGLISH - 5.2 */
 000003  0B TYPE(AB)
 000004  0D TYPE(PS)
 ..........................
 000201  0   Settings       Terminal and user parameters
 000202  1   View           Display source data or listings
 000203  2   Edit           Create or change source data
 000204  3   Utilities      Perform utility functions
 000205  4   Foreground     Interactive language processing
 000206  5   Batch          Submit job for language processing
 000207  6   Command        Enter TSO or Workstation commands
 ..........................
 000214  13 z/OS User       z/OS user applications
 000215  M   Minibank       Minibank Control Panel
 000216 )INIT
 000217 .ZVARS = '(ZCMD ZEXX)'
 ..........................
 000306    0,'PGM(ISPISM) SCRNAME(SETTINGS)'
 000307    1,'PGM(ISRBRO) PARM(ISRBRO01) SCRNAME(VIEW)'
 000308    2,'PGM(ISREDIT) PARM(P,ISREDM01) SCRNAME(EDIT)'
 000309    3,'PANEL(ISRUTIL) SCRNAME(UTIL)'
 000310    4,'PANEL(ISRFPA) SCRNAME(FOREGRND)'
 000311    5,'PGM(ISRJB1) PARM(ISRJPA) SCRNAME(BATCH) NOCHECK'
 000312    6,'PGM(ISRPTC) SCRNAME(CMD)'
 ..........................
 000319  13,'PANEL(ISR@390U) SCRNAME(OS390U)'
 000320    M,'cmd(exec ''minibank.panels.rexx(CP0000)'')'
 000321    X,EXIT
 000322  SP,'PGM(ISPSAM) PARM(PNS)'
 000323  ' ',' '
 000324    *,'?')
 000325 )PNTS
```

图 7-41 ISPF 主面板的定义

序在 ISPF 主面板定义一个入口。

上面介绍了如何修改 ISPF 主面板，为已经开发完毕的 ISPF 对话程序增加一个入口，类似地，也可以为不同的用户定制不同的 ISPF 主面板，这个案例留给读者自己去完成。

更多有关 ISPF 会话的有关内容，请参阅 IBM 白皮书 *ISPF Dialog Developer's Guide and Reference*。

7.6 REXX 与 ISPF 编辑宏

编辑宏是运行于 ISPF 编辑器环境下的 ISPF 会话。编辑宏是通常用由 CLIST 或者 REXX 等命令语言或者其他诸如 PL/I、COBOL 编写而成，使用这些编辑宏可扩展并定制编辑器命令。REXX 编辑宏就是一组完成特定编辑器任务的命令，存放在分区数据集的成员中，REXX 编辑宏可存放在包括 SYSUPROC、ALTLIB（专用于 REXX 程序的数据集）、SYSPROC 以及 SYSEXEC 对应的数据集中，并以 REXX 标志注释行/* REXX */（除 REXX 外也可包含其他字符，但 REXX 必不可少）作为首行。

编辑宏的使用和普通的编辑器命令相似，输入对应的宏名称，运行该编辑宏即可执行对应的编辑器相关功能。

7.6.1 编辑宏命令

在编辑宏中以 ISREDIT 语句开头的语句是编辑宏命令或赋值语句，这类语句交由对应的编辑器环境处理。可以使用 ADDRESS ISREDIT 命令切换 ISPF/PDF 编辑器宿主环境，所有后续命令将直接传给 ISPF/PDF 编辑器执行。也可以使用 ADDRESS ISPEXEC 命令切换到 ISPF 宿主环境，所有使用 ISREDIT+命令的方式执行编辑宏命令。表 7-11 列出了两种不同宿主环境中的使用方式。

表 7-11 2 个宿主环境

ISPEXEC 环境	**ISREDIT 环境**
ADDRESS ISPEXEC	*ADDRESS ISREDIT*
'ISREDIT EditMacroCommand/Assignment'	*EditMacroCommand/Assignment*

编辑器命令在 ISPF 编辑器中直接执行和在编辑宏 REXX 程序中执行效果基本相同，不同的是在 ISPF 命令行输入编辑器命令的运行结果会以消息或错误信息的方式显示给用户，而同样的命令在编辑宏 REXX 程序中执行则不会显示命令运行的具体信息，而是显示该宏执行的最终结果。此外需要注意一些特殊的命令需要附加操作符才能在宏中使用。所有的编辑宏必须以 ISREDIT MACRO 为第一条语句。

在 ISREDIT 之后除了可以输入编辑宏命令之外还可以直接输入赋值语句。

一般的编辑宏可通过 DEFINE 命令来完成定义。通过自定义编辑宏可以将现有的内置编辑命令替换为自定义的编辑宏。REXX 编辑宏是隐式定义的，即当用户使用一个非内置或者没有预定义名称的命令时，编辑器将自动搜索 SYSUEXEC、SYSUPROC、

ALTLIB、SYSEXEC 以及 SYSPROC 的库，如果这些库中含有一个该名称的成员，则将该命令自动识别为一个宏进行处理。

通过编辑宏的使用，用户可以：

- 完成重复性任务。对于重复性的编辑任务，可将其作为一个编辑宏存储，从而实现可多次利用，而不必重复敲击键盘。
- 简化复杂任务的处理。在完成复杂任务时直接使用包含处理逻辑的编辑宏，可大大简化复杂工作的处理步骤。
- 获取或返回信息。使用编辑宏可从编辑器或其他用户获取信息，并将处理后的消息返回给用户。

更多有关 ISPF 编辑宏的内容，可参阅 IBM 白皮书 *ISPF Edit and Edit Macros*。

7.6.2　编辑宏举例

通过在编辑宏中使用编辑赋值语句或命令语句，用户可以完成绝大多数在编辑器命令行所能实现的功能。例如移动、复制、插入等。对于文本编辑的行命令操作，基本都有对应的编辑宏命令可以用于完成这些文本操作。通过赋值语句也可以完成诸如光标定位等操作。

1. 复制行示例

如果需要复制一行，可通过将该行赋值给一个变量，然后通过插入行命令将保存的变量值赋给一个新行，即通过编辑赋值的方法操作行数据，从而完成复制行的功能。

```
/* REXX */
ADDRESS  ISPEXEC
'ISREDIT MACRO'
'ISREDIT (LINEDATA) = LINE 1'
'ISREDIT INSERT 5'
'ISREDIT LINE 5 = (LINEDATA) '
```

2. 光标定位示例

通过编辑宏把光标定位到第 3 行第 15 列。

```
/* REXX */
ADDRESS ISPEXEC
'ISREDIT MACRO'
'ISREDIT CURSOR = 3,15'
```

3. 删除特征行示例

实现编辑宏，将所有首列为‘-’符号的行删除，第一行除外。通过 ISREDIT 语句调用相应的编辑器命令（RESET/EXCLUDED/FIND/DELETE）来完成该功能。

```
/* REXX */
ADDRESS  ISPEXEC
```

```
"ISREDIT  MACRO"
"ISREDIT  RESET EXCLUDED "      /* Ensure no lines are excluded    */
"ISREDIT  EXCLUDE ALL '-' 1"    /* Exclude lines with '-' in col1  */
"ISREDIT  FIND FIRST '-' 1"     /* Show the first such line        */
"ISREDIT  DELETE ALL EXCLUDED"  /* Delete all lines left excluded  */
```

4. 增加虚线分割行示例

下例为一个较为复杂的完整示例，功能为将每行都用虚线分隔开。

```
/* REXX */
TRACE
ADDRESS  ISPEXEC
'ISREDIT  MACRO'
/*保存当前编辑配置信息、FIND/CHANGE 值以及面板和光标当前值*/
'ISREDIT  (SAVE) = USER_STATE'  /*USER_STATE*/
'ISREDIT  RESET'
'ISREDIT  EXCLUDE ----- 1 ALL'
'ISREDIT  DELETE ALL X'         /*删除原有的所有虚线行*/
LASTL = 1
LINE = 0
LINX = COPIES('-',70)
LL = LASTL + 1
DO WHILE LINE < LL
   'ISREDIT LINE_AFTER 'LINE' = (LINX)'
/*在 LINE 行后添加 LINX 变量的数据*/
   'ISREDIT (LASTL) = LINENUM .ZLAST'
    LL = LASTL + 1
    LINE = LINE + 2
END
'ISREDIT USER_STATE = (SAVE)'
EXIT
```

第 8 章

REXX 程序的执行

在主机系统中，通过 REXX 解释器直接解释执行或者由 REXX 编译器编译后通过编译运行器可直接运行的 REXX 语句，称为 REXX exec，即通常所说的 REXX 程序。

REXX 程序既可以通过解释的方式直接调用执行，也可以在编译后执行。本章首先介绍在不同环境下，如何通过解释的方式调用 REXX 脚本的方式；随后对执行编译后的 REXX 程序进行初步的介绍。

8.1　TSO/E 环境下 REXX 程序调用

在数据集完成了 REXX 程序的编写后，用户可以通过显式或隐式的方式在 MVS 的各个地址空间调用该 REXX 程序。这里的 REXX 程序调用，指的是通过 REXX 解释器直接解释执行 REXX 脚本。

1. 显式调用

在 TSO/E 下，显式使用 EXEC 命令，后跟包含 REXX 程序（exec）的数据集名称。其中，关键字 exec 是系统用于区分 REXX 与 CLIST 程序的参数，在调用 REXX 程序时，通常必须加上该参数。但是，如果在 REXX 脚本中的首行加入了/* REXX */注解，则在调用该脚本时，exec 参数可以省略。

根据 TSO/E 下数据集命名规范的不同，可使用下例对 USERID.REXX.EXEC(HELLO) 数据集的全名调用。

```
EXEC 'userid.rexx.exec(timegame)' exec
```

也可以不使用引号，使用省去用户前缀和后缀的调用方式非全名数据集调用，在这种情况下，系统会为数据集加上默认的前后缀，如下例所示，用户可在 TSO 环境下的 READY 提示符处输入命令（该处示例省略了数据集的前后缀）。

```
READY
EXEC rexx(timegame) exec
```

或者，用户还可以通过 ISPF/PDF 的 COMMAND 命令行输入命令，如图 8-1 所示，该命令省略了数据集用户前缀。

```
----------------------TSO COMMAND PROCESSOR ----------------------

ENTER TSO COMMAND OR CLIST BELOW:
===> exec rexx.exec (timegame) exec



ENTER SESSION MANAGER MODE ===> NO      (YES or NO)
```

图 8-1 REXX 程序执行

2. 隐式调用

隐式调用 REXX 程序（exec），即通过直接输入包含该 REXX 程序的分区数据集成员的名称，而直接调用该程序。使用隐式调用的前提是，这些分区数据集已经被设定为系统库（SYSPROC 或 SYSEXEC）。

因此，在隐式调用 REXX 程序之前，需要为对应的系统库（SYSPROC 或者 SYSEXEC）分配包含该 REXX 程序的分区数据集（PDS）。SYSPROC 与 SYSEXEC 的区别在于，在 SYSPROC 中同时存放 CLIST 程序和 REXX 程序（系统通过 exec 关键字标识以及 REXX 程序代码起始处所含有的/* REXX */首行注释来区分两者）；而在 SYSEXEC 中，只能存放 REXX 程序。在默认情况下，系统使用 SYSEXEC 作为 REXX 程序的默认系统库，因此，建议将日常使用或者重要的 REXX 程序放入 SYSEXEC 系统库中，使得这些脚本更易于维护与管理。关于 REXX 系统库的程序装载及搜索，可以参考 IBM 白皮书 *TSO/E REXX REFERENCE* 中的相关内容 *Using SYSPROC and SYSEXEC for REXX Execs*。

特别要说明的是，如果用户打算使用编译的方式来执行 REXX 程序，同样也可以将编译得到的 LOAD 模块放入对应的系统数据集中，并在执行时采用隐式调用。

使用 TSO/E 命令，可以为系统库（SYSPROC 或 SYSEXEC）分配一个 PDS 数据集，下面的例子通过执行的 TSO/E 命令，将数据集 userid.REXX.EXEC 以及 ISP.PHONE.EXEC 分配为 SYSEXEC 系统库。

```
EXECUTIL SEARCHDD(yes)
/* 验证系统库 SYSEXEC 是否可用*/
ALLOC FILE(SYSEXEC) DATASET(rexx.exec,'isp.phone.exec')
SHR REUSE
```

完成了对应 PDS 数据集的分配后，就可以在 READY 提示符处通过直接输入 REXX 程序所在分区数据集的成员名称来调用该 REXX 程序。

调用该 REXX 程序（exec）示例如下：

```
READY
Timegame
```

或者在 ISPF/PDF 的 COMMAND 命令行输入，如图 8-2 所示。或在任何的 ISPF/PDF 命令行前加 tso 后输入，如图 8-3 所示。

```
--------------------TSO COMMAND PROCESSOR --------------------

ENTER TSO COMMAND OR CLIST BELOW:
===> timegame


 ENTER SESSION MANAGER MODE ===> NO     (YES or NO)
```

图 8-2　REXX 程序调用

```
--------------------------------------------EDIT  - EDIT PANEL ---------------------------------------
COMMAND ===> tso timegame

ISPF LIBRARY :
   PROJECT  ===> PREFIX
   GROUP    ===> REXX    ===>          ===>          ===>
   TYPE     ===> EXEC
   MEMBER   ===> TIMEGAME      (Blank for member selection list)

OTHER PARTITIONED OR SEQUENCIAL DATA SET :
    DATA SET NAME      ===>
    VOLUME SERIAL      ===>    (If not cataloged)

    DATA SET PASSWORD  ===>    (If password protected)

    PROFILE NAME       ===>    (Blank default to data set type)

    INITIAL ,ACRO      ===>    LOCK         ===>  YES    (YES, NO or NEVER)

    FORMAT NAME        ===>    MIXED MODE   ===>  NO
```

图 8-3　ISPF 命令行中执行 REXX 程序

以上几种方式，均可完成对名为 timegame 的 REXX 程序的隐式调用。

如果在隐式调用 REXX 程序时，想要缩短搜索程序时间，则可以通过在程序名前加上符号%，这样系统会直接搜索 SYSEXEC 或 SYSPROC 库来查找对应的 REXX 程序。如果不加符号%，则系统会在搜索 SYSEXEC 和 SYSPROC 库之前，先遍历其他的系统库，以判断执行的是 TSO 命令还是 REXX 脚本。

另外，还可以通过压缩 REXX 程序的方式来提高执行性能，同时优化系统存储，其前提是该 REXX 程序被存放在 SYSPROC 库中，或者使用 ALTLIB 命令定义的 CLIST 应用级别程序库中。REXX 程序压缩的主要过程是删除注解、多余的空格、多余的空行，达到缩小程序体积的目的。

3. 后台调用

前面所述的显式和隐式调用均为在 TSO/E 前台调用。而通过批处理方式可在后台调用 REXX 程序（exec），使之自动运行而不需要人为干预。此种调用方法尤为适合运行优先级较低或者耗时长且不需要终端交互的 REXX 程序。

在后台运行 REXX 程序，可以使用 IJKEFT01 创建调用 TSO/E 命令/CLIST/REXX 所需的环境。例如，调用数据集 USERID.REXX 中名为 REXX1 的 REXX 程序（exec），可通过提交如下 JCL 完成：

```
// USERIDA JOB 'ACCOUNT, DEPT, BLDG','PROGRAMMER NAME',
```

```
// CLASS=J, MSGCLASS=C, MSGLEVEL= (1,1)
//TMP        EXEC PGM=IKJEFT01, DYNAMNBR=30, REGION=4096K
//SYSEXEC    DD   DSN=USERID.REXX, DISP=SHR
//SYSTSPRT   DD   SYSOUT=A
//SYSTSIN    DD   *
%REXX1
/*
```

上例中，EXEC 指定了执行实用程序 IJKEFT01。通过提交（SUBMIT）该作业，即可在后台调用 REXX1 程序，其执行结果如图 8-4 所示。

```
********************************** TOP OF DATA **********************************
        JES2 JOB LOG   SYSTEM SOW1   NODE

09.01.08 JOB06152 —— SUNDAY,  26 JUN 2011 ——
09.01.08 JOB06152 IRR010I USERID userid IS ASSIGNED TO THIS JOB.
09.01.09 JOB06152 ICH70001I userid LAST ACCESS AT 09:00:29 ON SUNDAY, JUNE 26
09.01.09 JOB06152 $HASP373 EXECREXX STARTED - INIT 1  - CLASS A - SYS SOW1
09.01.09 JOB06152 -                        ——TIMINGS (MINS.)—
09.01.09 JOB06152 -STEPNAME PROCSTEP   RC  EXCP  CONN    TCB    SRB  C
09.01.09 JOB06152 -TMP                 00   15    10     .00    .00
09.01.09 JOB06152 -EXECREXX ENDED.  NAME-             TOTAL TCB CPU TIM
09.01.09 JOB06152 $HASP395 EXECREXX ENDED
—— JES2 JOB STATISTICS ——
 26 JUN 2011 JOB EXECUTION DATE
       7 CARDS READ
      46 SYSOUT PRINT RECORDS
       0 SYSOUT PUNCH RECORDS
       3 SYSOUT SPOOL KBYTES
    0.00 MINUTES EXECUTION TIME
     1 //EXECREXX JOB NOTIFY=&SYSUID
       IEFC653I SUBSTITUTION JCL - NOTIFY=userid
     2 //TMP     EXEC PGM=IKJEFT01
     3 //SYSEXEC  DD DSN=userid.REXX,DISP=SHR
     4 //SYSTSPRT DD SYSOUT=A
     5 //SYSTSIN  DD *
ICH70001I userid  LAST ACCESS AT 09:00:29 ON SUNDAY, JUNE 26, 2011
IEF236I ALLOC. FOR EXECREXX TMP
IEF237I 0D30 ALLOCATED TO SYSEXEC
IEF237I JES2 ALLOCATED TO SYSTSPRT
IEF237I JES2 ALLOCATED TO SYSTSIN
IEF142I EXECREXX TMP - STEP WAS EXECUTED - COND CODE 0000
IEF285I   userid.REXX                          KEPT
IEF285I   VOL SER NOS= VPWRKA.
IEF285I   userid.EXECREXX.JOB06152.D0000102.?      SYSOUT
IEF285I   userid.EXECREXX.JOB06152.D0000101.?      SYSIN
IEF373I STEP/TMP     /START 2011177.0901
IEF032I STEP/TMP     /STOP  2011177.0901
     CPU:   0 HR  00 MIN  00.01 SEC   SRB:   0 HR  00 MIN  00.00 SEC
     VIRT:   48K  SYS:  344K  EXT:     328K  SYS:   12036K
IEF375I  JOB/EXECREXX/START 2011177.0901
IEF033I  JOB/EXECREXX/STOP  2011177.0901
     CPU:   0 HR  00 MIN  00.01 SEC   SRB:   0 HR  00 MIN  00.00 SEC
READY
 %REXX1
THIS IS A REXX PROGRAM
READY
END
********************************* BOTTOM OF DATA *********************************
```

图 8-4　REXX 程序后台执行结果查看

4. 其他调用

前文所述内容均为在 TSO/E 环境（地址空间）下的 REXX 基本调用方式。除此之外，还可以通过 CLIST 脚本，达到输入相应命令调用程序运行的目的（由于 CLIST 脚本就是作为 TSO/E 环境下的命令序列）。

在此指出，无论使用以上何种方式调用 REXX 程序（exec），都可以在调用命令行中为被调 REXX 程序传递参数。

下面的例子中，执行名为 add 的 REXX 程序（exec）时，需要传入两个加法运算参数 42 和 21，可以通过如下方式：

```
EXEC rexx.exec(add) '42 21' exec
```

关于 REXX 程序如何接受参数、与终端如何交互以及输入输出处理等议题，将在本章后面的部分陆续涉及。

注意：这里所涉及的调用方法都是在 TSO/E 地址空间下的，事实上，如果要在非 TSO/E 地址空间下调用 REXX 程序，则不能使用 TSO/E 命令，需要用别的方法替代。

更多相关信息，可参阅 IBM 白皮书 *TSO REXX User's Guide*。

8.2　非 TSO/E 环境下 REXX 程序调用

在非 TSO/E 环境下，使用 TSO/E EXEC 命令来调用 REXX 程序已不再可行，所以必须采用其他的方式。在非 TSO/E 的地址空间中调用 REXX 程序，以下两种方法都可以实现这个目的。

- 前台调用，在高级别程序中使用 IRXEXEC 或 IRXJCL 处理例程（Processing Routine）。
- 后台调用，在 JCL 中将 EXEC 语句指定的程序（PGM）设置为 IRXJCL。

此外，TSO/E 提供的环境服务（Environment Service）IKJTSOEV，可以在一个非 TSO/E 的地址空间中创建一个 TSO/E 地址空间。在这个被新创建出来的地址空间中，用户可以自由使用 TSO/E 命令（包括了 EXEC 命令）、TSO/E 外部函数，以及其他在 TSO/E 环境中可以被 REXX 程序调用的服务。

关于 TSO/E 环境服务的详细信息，请参考 IBM 白皮书 *z/OS TSO/E Programming Services*。

1. 前台调用

IRXEXEC 和 IRXJCL 都是 TSO/E 为 REXX 编程提供的服务例程，它们作为 REXX 语言处理器的编程接口，为用户提供了在 MVS 环境下的任何地址空间中，调用 REXX 程序的渠道。

如果想通过在 MVS 地址空间中运行的高级语言程序来调用某个 REXX 程序，可以使用 IRXEXEC 或 IRXJCL 处理例程。如果使用 IRXEXEC，需要预先设置好被调用 REXX

程序的相关参数以及其他信息。

下面是使用 PL/I 程序以调用 IRXJCL 从而执行一个 REXX 程序的例子。

```
JCLXMP1 : Procedure Options (Main);
/* Function: Call a REXX exec from a PL/I program
                                             using IRXJCL */
DCL IRXJCL EXTERNAL OPTIONS(RETCODE, ASSEMBLER);
DCL 1 PARM_STRUCT,
                        /* Parm to be passed to IRXJCL */
    5 PARM_LNG BIN FIXED (15),
                        /* Length of the parameter */
     5 PARM_STR CHAR (30);
                        /* String passed to IRXJCL */
DCL PLIRETV BUILTIN;
                        /* Defines the return code built-in*/
PARM_LNG = LENGTH(PARM_STR);
                        /* Set the length of string */
/*                                                          */
PARM_STR = 'JCLXMP2 This is an arg to exec';
/* Set string value In this case, call the exec
                        named JCLXMP2 and pass argument:
                                'This is an arg to exec' */
FETCH IRXJCL;
                        /* Load the address of entry point */
CALL IRXJCL (PARM_STRUCT);
/* Call IRXJCL to execute the REXX exec and pass
                                            the argument */
PUT SKIP EDIT ('Return code from IRXJCL was:', PLIRETV) (a, f(4));
/* Print out the return code from exec JCLXMP2. */
END;                                 /* End of program */
```

在调用 IRXJCL 前，需要先将参数列表的地址传入到寄存器 1（Register 1）中。如果 IRXJCL 的调用遇到了错误，则会将返回码 RC 传入寄存器 15 中，以便于用户查明错误原因。

使用 IRXEXEC 调用 REXX 程序，相对于使用 IRXICL 来调用能够带来更大的灵活性。例如，当用户想反复多次执行一个 REXX 脚本时，可以将该脚本预先装载到内存中，然后将其地址传递给 IRXEXEC。这样，就避免了在调用 REXX 脚本时系统频繁地载入内存和清空内存的额外消耗。

其实，在 TSO/E 环境下，IRXEXEC 和 IRXJCL 处理例程也常常用来调用 REXX 程序。因为在显式或隐式调用 REXX 程序的同时只能传递一个变量字符串，而通过 IRXEXEC 则可以传递多个变量字符串。另外，对于非 REXX 或 CLIST 的程序（如 COBOL 程序），如果想要调用 REXX 程序，则必须要用到这两个处理例程。

关于 IRXEXEC 和 IRXJCL，以及 TSO/E 编程服务的详细信息，请查看 IBM 白皮书

TSO REXX Reference。

2. 后台调用

如果想在 MVS 批处理环境下调用一个 REXX 程序，必须在 JCL 的 EXEC 语句中，将 PROGRAM 参数设定为 IRXJCL。

在 MVS 环境下对 REXX 程序的后台调用与在 TSO/E 环境下后台调用 REXX 程序非常相似，但也有一些不同之处，其中最大的区别在于，MVS 环境下调用的 REXX 程序无法使用 TSO/E 命令、TSO/E 外部函数等功能。

下面的例子在 MVS 环境下，后台调用 REXX 程序 USERID.MYREXX.EXEC (JCLTEST)，而作业的执行输出则存入了数据集 USERID.IRXJCL.OUTPUT 中。

其中，执行作业 USERID.JCL.EXEC 的内容如下。

```
//USERIDA JOB NOTIFY=&SYSUID
//MVSBACH EXEC PGM=IRXJCL,
// PARM='JCLTEST Test IRXJCL'
//OUTDD DD DSN=USERID.TRACE.OUTPUT,DISP=MOD
//SYSTSPRT DD DSN=USERID.IRXJCL.OUTPUT,DISP=OLD
//SYSEXEC DD DSN=USERID.MYREXX.EXEC,DISP=SHR
//SYSTSIN DD *
First line of data
Second line of data
Third line of data
/*
//
```

在该作业中，PARM 参数对应的字符串的第一部分（也即 JCLTEST），指定的是被调用 REXX 程序的名称；而剩下的部分，则指定了传给该程序的参数变量（在 REXX 程序中通过 ARG 语句接收这些参数）。

被调用的 REXX 程序 USERID.MYREXX.EXEC(JCLTEST)，其代码如下。

```
/************************** REXX **************************/
      /* This exec receives input from its invocation in  */
                                   /* JCL.EXEC, pulls */
/* data from the input stream and sends back a condition  */
                                     /* code of 137.  */
/*******************************************************/
TRACE error
SAY 'Running exec JCLTEST'
ADDRESS MVS
PARSE ARG input
SAY input
DATA = start
DO UNTIL DATA = '
PARSE PULL data          /* pull data from the input stream*/
```

```
SAY data
END
/**********************************************************/
/* Now use EXECIO to write a timestamp to the sequential   */
/* data set that was allocated to the OUTDD file by the JCL*/
                              /* used to invoke this exec.*/
/**********************************************************/
OUTLINE.1 = 'Exec JCLTEST has ended at' TIME()
"EXECIO 1 DISKW OUTDD (STEM OUTLINE. FINIS"
                                         /* Write the line*/
SAY 'Leaving exec JCLTEST'
EXIT 137                  /* send a condition code of 137 */
```

这个 REXX 程序首先读取并输出 JCL 中 PARM 参数传递的变量，随后读取并输出 JCL 中 SYSTSIN 输入流中的信息。接着，该程序将一个时间戳（Timestamp）写入到了 JCL 中指定的输出数据集 USERID.TRACE.OUTPUT 中。最后，该程序返回 Condition Code 137。

时间戳的输出数据集 USERID.TRACE.OUTPUT 中的内容如下。

```
Exec JCLTEST has ended at 15:03:06
```

而存放在数据集 USERID.IRXJCL.OUTPUT 中的作业执行输出结果内容如下。

```
Running exec JCLTEST
Test IRXJCL
First line of data
Second line of data
Third line of data
Leaving exec JCLTEST
```

最后，可以看到整个作业执行过程的部分信息如下。

```
ALLOC. FOR USERIDA MVSBACH
224 ALLOCATED TO OUTDD
954 ALLOCATED TO SYSTSPRT
7E0 ALLOCATED TO SYSEXEC
JES2 ALLOCATED TO SYSTSIN
USERIDA MVSBACH - STEP WAS EXECUTED - COND CODE 0137
USERID.TRACE.OUTPUT KEPT
VOL SER NOS= TSO032.
USERID.IRXJCL.OUTPUT KEPT
VOL SER NOS= TSO032.
USERID.MYREXX.EXEC KEPT
VOL SER NOS= TSO001.
JES2.JOB28359.I0000101 SYSIN
STEP / MVSBACH / START 88167.0826
STEP / MVSBACH / STOP 88167.0826 CPU 0MIN 00.16SEC SRB ...
```

```
JOB / USERIDA / START 88167.0826
JOB / USERIDA / STOP 88167.0826 CPU 0MIN 00.16SEC SRB ...
```

8.3　REXX 程序的编译

通过使用 REXX 370 编译器，可以在 z/OS 以及 z/VM 环境下对 REXX 程序进行编译。实际上 REXX 编译器由 3 部分组成，它们分别是：

- REXX 编译器（IBM Compiler for REXX）：该编译器完成将 REXX 源程序转化为可执行模块的工作。
- REXX 库（IBM Library for REXX）：该程序库提供在 REXX 可执行模块运行时需要调用的例程（Routine）。
- REXX 备用库（IBM Alternate Library for REXX）（可免费下载）：该程序库可以将 REXX 可执行模块转化为 REXX 解释器可以识别的形式。这样的话，没有安装 REXX 库的系统，也能够执行 REXX 可执行模块。

关于 REXX 程序编译的详细信息，可以查看 IBM 白皮书 *IBM Compiler and Library for REXX on zSeries User's Guide and Reference*。

1. 编译 REXX 程序

接下来的例子将编译并执行本书的第一个 REXX 程序：Hello World（见第 2 章）。使用下面这条命令可以完成对该程序的编译。

```
REXXC 'userid.REXX(HELLO)' CEXEC('userid.CREXX(HELLO)')
```

其中，REXX 源程序为 userid.REXX(HELLO)，而编译后的 REXX 目标可执行模块将为 userid.CREXX(HELLO)。执行该命令的结果如下。

```
REXXC starting
Compiling     'userid.REXX(HELLO)'
PRINT output  'userid.REXX.HELLO.LIST'
CEXEC output  'userid.CREXX(HELLO)'
Compiler return code is 0
***
```

可以看到，编译过程中的相关信息被存放在系统自动生成的顺序数据集 userid.REXX.HELLO.LIST 中。进入该数据集，可以查看这些信息。另外，在执行命令时可以指定 PRINT 参数，将编译过程的相关信息如顺序数据集直接打印到终端上而不存放，其命令如下。

```
REXXC 'userid.REXX(HELLO)'
CEXEC('userid.CREXX(HELLO)') PRINT(*)
```

该命令的执行结果部分信息如下。

```
REXXC starting
Compiling    'userid.REXX(HELLO)'
PRINT output  to terminal
CEXEC output  'userid.CREXX(HELLO)'
===> Compilation Summary
IBM100S.REXX(HELLO)
IBM Compiler for REXX on zSeries 4.0  LVL PK04822   Time: 20:14:10   Date:
2011-06-26                  Page:    1
Compilation successful
===> Compilation Statistics
userid.REXX(HELLO)
IBM Compiler for REXX on zSeries 4.0  LVL PK04822   Time: 20:14:10   Date:
2011-06-26                  Page:    3
REXX Lines  6
Size of compiled program in bytes  3152
Finishing time of compilation:  20:14:10
Compiler return code is 0
***
```

随后，运行编译好的 REXX 可执行模块，其结果如图 8-5 所示。

```
  Menu  Functions  Confirm  Utilities  Help
----------------------------------------------------------------------------
EDIT              IBMUSER.CREXX                              Row00001 of 00003
Command ===>                                                    Scroll ===> PAGE
           Name       Prompt       Size    Created           Changed         ID
__EXEC___  HELLO
_________  PERFORM
_________  PERFORM2
           **End**




 WHAT'S YOUR NAME?
USER
 HELLO WORLD! THIS IS USER
 ***
```

图 8-5　REXX 程序执行结果

2. 编译 REXX 程序性能测试

在第 2 章中已经提到，对 REXX 程序进行编译，可以提升性能、降低系统装载量、保护源代码安全、提升编程效率和质量，以及提升编译后程序的可移植性等。事实上，通过对于执行 REXX 源程序以及 REXX 编译模块的性能比较，确实可以发现，执行编译后的 REXX 程序的性能要高出许多。

首先针对字符串处理进行测试，REXX 脚本代码如下。

```
/* REXX */
```

```
PARSE VERSION v
SAY v
N = TIME('E')
DO I =1 TO 5000000
     A = 'STRING'
     N = I || A
END
SAY TIME('E')
```

通过 REXX 解释器，执行该 REXX 脚本，其输出结果如下。

```
REXX370 3.48 01 MAY 1992
11.693426
```

即以解释的方式执行该脚本，需要花费的时间为 11.693426 秒。而执行该 REXX 脚本的编译模块，其输出结果如下。

```
REXXC370 3.48 23 DEC 1999
1.196127
```

即以编译的方式执行该脚本，需要花费的时间仅为 1.196127 秒，不到直接执行该脚本所需花费时间的 10%，可以说，性能有了大幅度的提升。

接下来是针对算术操作进行测试，REXX 脚本代码如下。

```
/* REXX */
PARSE VERSION v
SAY v
N = TIME('E')
DO I =1 TO 1000000
     N = I * I
END
SAY TIME('E')
```

同样通过 REXX 解释器，执行该 REXX 脚本，其输出结果如下。

```
REXX370 3.48 01 MAY 1992
3.491962
```

而执行这第二个 REXX 脚本的编译模块，其输出结果如下。

```
REXXC370 3.48 23 DEC 1999
0.230007
```

可见，进行算术操作时，两种 REXX 程序执行方式的性能差距甚至更为巨大。在通常情况下，根据程序的类型不同，执行编译后的 REXX 程序会带来的性能提升的预期结果如表 8-1 所示。

其中，性能提升幅度最大的是执行大量算术操作；而性能提升幅度最小的是执行宿

主命令，因为编译器无法减少宿主环境执行这些命令的时间。

表 8-1　REXX 程序编译执行优势

程序主要操作	编译后运行速度提高的倍数
算术运算（arithmetic operations）	6～10 倍
字符串处理（string and word processing operations）	
常量和变量操作（constants and variables）	4～6 倍
调用过程及内置函数（references to procedures and built-in functions）	
修改变量值（changes to values of variables）	
赋值操作（assignments）	2～4 倍
重用复合变量（reused compound variables）	
宿主命令（host commands）	无显著提升
文件 I/O	

第 9 章 REXX 程序的调试

一个好的程序，需要具有一定的可靠性、易维护性，能够考虑到特殊情况的处理以及给用户提示出错信息等。在第 2 章中，已经简单介绍了 REXX 程序的调试方法，本章将详细讲解这些命令的使用方法以及调试工具的操作。

9.1 异常情况的跟踪

异常情况是指在 REXX 中，当使用了 CALL ON 或 SIGNAL ON 时，可以被跟踪的事件或状态，通过对一个特定情况的跟踪，可以改变程序的执行顺序，具体的用法如下。

```
CALL/SIGNAL ON/OFF condition (trapname)
```

首先需要通过 ON 或 OFF 来指定是否开启对特定情况的跟踪工作。在默认情况下，所有的跟踪都是关闭的。如果开启了对特定事件的跟踪，那么一旦特定事件发生了，程序会跳转到对应的处理位置（trapname 处）继续执行。在 CALL 指令中，trapname 可以是内部标签、内置函数或外部例程的名字，但是 SIGNAL 指令中，trapname 只能是程序内部的标签。另外，每一个 CALL/SIGNAL ON/OFF 指令都将取代之前的设置。

9.1.1 事件分类

被跟踪的事件（Condition）有 5 种：Error、Failure、Halt、Novalue、Syntax。

（1）ERROR，指如果命令产生了 error；或者是命令 failure 但并未设置 CALL ON FAILURE 或 SIGNAL ON FAILURE，则在该命令调用的返回处进行事件处理。但是如果在此之前，已经有 ERROR 的事件延迟处理（处于 Delay 状态），将忽略此处的 ERROR 事件。

在 TSO/E 环境下，SIGNAL ON ERROR 指令跟踪所有命令返回码为正的情况，以及那些未被 CALL/SIGNAL ON FAILURE 跟踪的命令返回码为负的情况。

```
SIGNAL ON ERROR
"LISTSD DA(TE02.DB2.SRC)"   /* 错误的 TSO/E 命令 */
EXIT
ERROR:
```

```
SAY "ERROR IS RAISED"        /* RESULT */
```

（2）FAILURE，指如果命令产生了 failure 的结果，将在命令执行返回处被跟踪。如果此前有延迟处理的 FAILURE 事件，此处的事件将被忽略。在 TSO/E 环境下，CALL/SIGNAL ON FAILURE 指令跟踪所有命令返回码为负的情况。

```
SIGNAL ON FAILURE
CALL FUNCADD("Function1", "myOpe", "Function2")
/* 如果 FUNCADD 函数执行失败,将跳转到 FAILURE 处理程序 */
EXIT.
FAILURE:
SAY "FAILURE IS RAISED."     /* RESULT */
```

（3）HALT，指如果有外部命令（如 PA1）试图中断或终止程序的执行，将在被中止的语句结束处开始跳转到 HALT 事件处理。如在 TSO/E 环境下，HI 和 EXECUTIL HI 会引起 HALT 事件，HE 则不会引起此事件。

```
/* REXX */
SIGNAL ON HALT
DO I=1 TO 10000              /* PA1 pressed and HI enterred */
   SAY HELLO
END
EXIT
HALT:
SAY "HALT IS RAISED."        /* Result */
```

（4）NOVALUE，指如果未初始化的变量用在表达式中，或用在 PARSE VAR 之后的 PROCEDURE 或 DROP 指令中时，将被跟踪。注意，SIGNAL ON NOVALUE 指令将对除了复合变量尾部之外的所有未初始化的变量进行跟踪。

```
SIGNAL ON NOVALUE
DROP VAR
SAY VAR                      /* 触发 NOVALUE */
SAY 'NOVALUE IS NOT RAISED.'
EXIT
NOVALUE:
SAY 'NOVALUE IS RAISED.'     /* RESULT */
```

（5）SYNTAX，指在编译过程中如果出现了错误，包括语法错误和运行时错误。如对非数字进行算术运算等，该事件仅对 SIGNAL 指令有效。

```
/* REXX */
SIGNAL ON SYNTAX
SAY A+B                      /* SYNTAX ERROR */
SAY "SYNTAX ERROR IS NOT RAISED."
EXIT
SYNTAX:
```

```
SAY "SYNTAX ERROR IS RAISED."   /* RESULT */
```

9.1.2　事件处理

如果当前对事件的跟踪被关闭，而特定的事件又发生了，那么对于 HALT 和 SYNTAX 事件，程序将终止执行，同时会给用户反馈一条表明发生事件的提示信息，其他事件将不做处理。

```
DROP VAR
SAY VAR                         /* NOVALUE 事件,正常执行,输出 VAR */
EXIT

NEWDATE = DATE('F')
SAY NEWDATE                     /* SYNTAX 事件,不能执行 */
EXIT
/* 报错信息如下 */
2 +++ NEWDATE = DATE('F')
Error running TEST, line 2: Incorrect call to routine
```

如果当前对事件的跟踪被开启，而特定的事件又发生了，那么将有对应的子程序处理。如果是由 SIGNAL 指令进行的跟踪，那么处理之后，该跟踪将被关闭。如果想继续对同样的事件进行跟踪，则需要再次使用 SIGNAL/CALL ON。

如果是由 CALL 指令进行的跟踪，那么将按照正常的 CALL 指令流程执行，但是不对 RESULT 赋值。同时，如果这些事件发生在 INTERPRET 指令执行期间，那么 INTERPRET 指令将被终止，在 RETURN 之后继续执行。CALL 指令并不会影响原本程序的执行顺序。

在交互式跟踪模式下，当处理用户的输入信息时，对事件的跟踪状态将被暂时关闭。这样做是为了防止程序的意外跳转，同样，任何语法错误都不会使程序退出，而是给出一个提示信息。

9.1.3　事件信息

当开启对某个事件的处理，相关的信息将被记录下来。可以用 CONDITION 内置函数来查看，包括事件名称、跟踪指令（CALL 或 SIGNAL）、事件的状态以及该事件相关的描述信息等。

当程序跳转到跟踪事件处理程序部分，系统开始记录相关的事件信息，同时这些信息在函数和子例程的调用过程中将被保存，因此可以使用 CALL ON 命令来查看当前的跟踪信息，在该命令返回后，之前的跟踪信息也可以查询到。

记录信息的内容如下。

- 对于 ERROR 和 FAILURE 事件，REXX 程序会记录引发事件的语句。
- 对于 HALT 事件，REXX 程序会记录和 HALT 事件相关的信息。

- 对于 NOVALUE 事件，记录引发事件的变量名称。
- 对于 SYNTAX 事件，记录编译时引发事件的语句，可以为空，通过特殊变量 RC 和 SIGL 可以得到更具体的编译错误信息。

示例如下。

```
/* REXX */
SIGNAL ON SYNTAX
A = A + 1                              /* SYNTAX ERROR */
SAY 'SYNTAX ERROR NOT RAISED'
EXIT
SYNTAX:
SAY "CONDITION NAME:" CONDITION('C')   /* CONDITION NAME: SYNTAX */
SAY "INSTRUCTION:" CONDITION('I')      /* INSTRUCTION: SIGNAL    */
SAY "STATUS:" CONDITION('S')           /* STATUS: OFF            */
SAY "DESCRIPTION:" CONDITION('D')      /* DESCRIPTION:           */
```

9.2 诊断函数的使用

在调试程序的时候，可以使用 REXX 内置函数来帮忙查找相关的出错信息，它们是 SOURCELINE()、CONDITION()和 ERRORTEXT()。这 3 个函数是在诊断 REXX 程序时常用的函数，在第 4 章已经简单介绍过，下面着重介绍用它们来调试程序的方法。

1. REXX 内置函数

（1）SOURCELINE 函数

SOURCELINE 函数用于返回程序源代码，使用方法是 SOURCELINE(*n*)，其中 *n* 指定要查看的代码所在的行数。如果未指定 *n* 的值，则返回源程序代码的行数；如果对源程序没有访问权限，则返回 0；如果无法读取到第 *n* 行的数据，就返回空字串。注意，*n* 必须是一个正整数且不能超过源代码的行数范围。

```
/* REXX */
SAY HELLO WORLD
SAY SOURCELINE()    /* return lines of source code : 4 */
SAY SOURCELINE(2)   /* SAY HELLO WORLD */
```

（2）CONDITION 函数

CONDITION 函数返回当前跟踪事件的相关信息，通过以下可选参数来选择返回的信息类别。

① Condition name，返回当前跟踪事件的名称。

```
CONDITION('C') -> 'ERROR'
```

② Description，返回当前跟踪事件的相关描述信息，详见 9.4.3 节。

```
CONDITION('D') -> 'SIGINT'
```

③ Instruction，返回 CALL 或 SIGNAL，即发起事件跟踪的指令。

```
CONDITION('I') -> 'CALL'
```

④ Status，返回当前被跟踪事件的状态，可以是 ON、OFF 和 DELAY，ON、OFF 分别表示对该事件的跟踪已开启、关闭，DELAY 则表示后续新的同类型事件将被忽略。

```
CONDITION('S') -> 'OFF'
```

（3）ERRORTEXT 函数

ERRORTEXT 函数返回相应的 error 信息，格式为 ERRORTEXT(*n*)，其中 *n* 指定了错误代码，在 0～99 之间。错误代码是在 REXX 程序执行期间产生的，并且当指定 SIGNAL ON SYNTAX 时，存放在特殊变量 RC 中。

```
ERRORTEXT(7) -> ' WHEN or OTHERWISE expected '
ERRORTEXT(6) -> ' Unmatched "/*" or quote '
```

2. 特殊变量 RC 和 SIGL

在 REXX 语言中，共有 3 个特殊变量，分别是 RC、RESULT 和 SIGL。特殊变量没有初始值，可以直接使用。其中 RC 和 SIGL 可以用于程序的调试。

（1）特殊变量 RC

在 ERROR、FAILURE 和 SIGNAL SYNTAX 事件中，系统会在程序跳转到事件处理程序前将相应命令的返回码或对应语法错误的序号赋给 RC 变量，HALT 和 NOVALUE 事件不影响 RC 的值，在调试模式下手动输入的宿主命令也不影响 RC 的值。

```
/* REXX */
"LISTDS ?"
SAY "RC:" RC   /* 12 */
```

（2）特殊变量 SIGL

在 CALL 和 SIGNAL 指令跳转之后，系统将跳转之前最后被执行的语句行数存储在 SIGL 中，可以用来帮助判断错误的发生原因，如下例，在事件处理程序中，输出了错误代码、发生的行数和错误信息，然后得到了发生错误的具体代码。

```
SIGNAL ON SYNTAX
A = A + 1 /* 产生 SYNTAX ERROR */
SAY 'SYNTAX ERROR NOT RAISED'
EXIT
/* 标准 SIGNAL ON SYNTAX 处理程序 */
SYNTAX:
SAY 'REXX ERROR' RC 'IN LINE' SIGL':' ERRORTEXT(RC)
SAY SOURCELINE(SIGL)
```

9.3 程序异常处理示例

在了解了 REXX 的异常事件处理及诊断函数之后，来看两个具体的案例。

案例一：NOVALUE 异常处理。通过下面的 NOVALUE 异常处理代码，程序 REXXSAMP 出现变量未初始化的异常时，系统将返回更详细的错误信息，以帮助用户定位问题变量，及时更正。

```
/* REXX REXXSAMP source code */
SIGNAL ON NOVALUE
Say VAR
EXIT
/* NOVALUE EVENT HANDLER */
NOVALUE:
Trace N
Do While Queued()<>0  /* Clear the data stack */
    Pull
End
Say "EXEC REXXSAMP: Uninitialized variable used."
Say "Name of uninitialized variable: " Condition('D')
Say "Problem line number=" sigl
Say "Problem line content: "Sourceline(sigl)
EXIT

/* REXXSAMP 运行结果 */
EXEC REXXSAMP: Uninitialized variable used.
Name of uninitialized variable:  VAR
Problem line number= 3
Problem line content: Say VAR
```

案例二：ERROR 异常处理。当程序 REXXSAMP 出现命令执行错误时，通过下面的 ERROR 异常处理代码，REXX 程序将返回更详细的错误信息，以帮助用户定位问题命令，及时更正。

```
/* REXX REXXSAMP source code */
SIGNAL ON ERROR
"LISTSD"
EXIT
/* ERROR EVENT HANDLER when non zero rc from the host environment */
ERROR:
Signal off Error
Signal off Novalue
Trace N
Do While Queued()<>0  /* Clear the data stack */
```

```
    Pull
End
Say "EXEC REXXSAMP: Host command failed or data format error."
Say "RC =" rc
Say "Error line number = " sigl
Say "Error line content = " Sourceline(sigl)
Say "Error message text: " Condition('D')
EXIT

/* REXXSAMP 运行结果 */
EXEC REXXSAMP: Host command failed or data format error.
RC = -3
Error line number =  3
Error line content =  "LISTSD"
Error message text:  LISTSD
```

9.4　使用 Trace 指令

Trace 指令用来控制程序的跟踪流程，显示程序执行时的语句输出。TRACE 指令通常用于调试，可以实现交互式的调试功能。TRACE 指令的格式如下。

```
TRACE(number)
TRACE(?/!+ 'Normal/All/Commands/Error/Failure/Intermdiates/Labels/Off/
Results/Scan')
TRACE(string/symbol/expression)
```

下面将介绍这 3 种指令格式的具体用法。

9.4.1　字母参数

在 TRACE 函数中，可以指定跟踪的内容和方式，具体的参数如下，程序中可以只用大写的首字母代表。

- All：跟踪所有语句。
- Commands：显示所有被跟踪的命令，如果命令执行结果有误，再输出命令的返回值。
- Error：跟踪那些执行后出错（error 或 failure）的命令，即返回码非 0 的命令，并记录它们的返回码。
- Failure：跟踪那些造成系统错误（failure）的命令，即返回码为负的命令，并记录返回码。
- Intermediates：跟踪所有语句，并且记录生成表达式和名称替换等中间执行结果。
- Labels：跟踪标签的执行，特别适合在调试模式下使用。帮助用户跟踪所有内部子例程的调用以及 Signal 指令对程序控制权的转换。

- Normal：跟踪那些执行后返回码为负的语句以及它们的返回码，是默认类型。
- Off：不跟踪语句，并且恢复特殊前缀的默认属性。
- Results：跟踪全部语句，并且输出表达式转换的最终结果（非中间结果），以及 PULL、ARG、PARSE 指令的赋值结果，通常在调试时使用。
- Scan：跟踪所有未处理的数据，通常进行一些基本格式的检查（如缺少 END）等，前提是 TRACE S 命令本身没有被嵌套在指令中。

下面是一些对应内容的跟踪示例。

（1）A 选项的跟踪过程，输出程序指令，并显示执行输出。

```
/* REXX source code */
TRACE 'A'
QUEUE 'some data'
PULL MyVar
SAY MyVar
EXIT

/* 跟踪输出 */
     3 *-* QUEUE 'some data'
     4 *-* PULL MyVar
     5 *-* SAY MyVar
SOME DATA
     6 *-* EXIT
```

（2）在 R 选项的跟踪过程，与 A 相似，但是显示了参数的传递过程。

```
     3 *-* QUEUE 'some data'
       >>>   "some data"
     4 *-* PULL MyVar
       >>>   "SOME DATA"
     5 *-* SAY MyVar
       >>>   "SOME DATA"
SOME DATA
     6 *-* EXIT
```

（3）在 L 模式下，跟踪标签的执行。

```
/* REXX source code */
TRACE 'L'
DO i = 1 TO 10
   CALL Sub
   IF i = 3 THEN SIGNAL jump
END
jump:
EXIT
Sub:
   RETURN
```

```
/* 跟踪输出 */
9 *-*   Sub:
  *-*   Sub:
  *-*   Sub:
7 *-* jump:
```

（4）在 C 模式下，显示传送给外部环境的命令字符串。下例中命令以外的语句，如 A=1、B=2 都不会被跟踪，只有命令 LU 被跟踪。

```
/* REXX source code */
TRACE 'C'
A=1;
USER=USERID()
'LU 'USER
B=2

/* 跟踪输出 */
5 *-* 'LU 'USER
  >>>   "LU TE02"
```

（5）在 I 模式下，输出执行时的详细处理信息。

```
TRACE 'I'
MyVar = 'This is some data'
PARSE VAR MyVar token1. token2.
EXIT

/* 跟踪输出 */
3 *-* MyVar = 'This is some data'
  >L>   "This is some data"
4 *-* PARSE VAR MyVar token1 . token2 .
  >>>   "This"
  >.>   "is"
  >>>   "some"
  >.>   "data"
5 *-* EXIT
```

9.4.2 前缀参数

TRACE 指令有两个前缀？和!，可以单独使用，也可以和字母选项共同使用，如果和字母一起使用，注意在两者中间不能插入空格。

！前缀用于停止宿主命令的执行（但是可以调试），如 TRACE !C 指令使得所有命令被跟踪但是不被执行，即跳过这些命令，因为所有的命令都没有真正执行，所以 RC 将会是 0。这个前缀通常用于调试某些特殊的有潜在危险的 REXX 程序，来避免调试过程

中 REXX 程序对主机环境造成破坏。输入一次 TRACE !可以进入该模式，而再次输入 TRACE !即可关闭该模式。在调试的任意时刻输入 TRACE O，或者输入 TRACE，也可以关闭该模式。注意，在交互调试模式下输入的宿主指令，永远会被真正执行，不受!前缀指令的影响，但是不会对 RC 赋值。

```
"NEWSTACK"
"NEWSTACK"
"NEWSTACK"
"QSTACK"
TIMES = RC-1      /* TIMES = 3 */
DO TIMES
TRACE !C
"DELSTACK"        /* 该命令并未执行 */
END
TRACE O           /* 关闭调试 */
"QSTACK"          /* RC = 4 */
```

调试输出：

```
10 *-* 'DELSTACK'
```

? 前缀用于控制交互式调试，如 TRACE ?E 指令使得编译器在每次命令执行出错时暂停等待用户输入。注意，当使用了交互调试后，程序内原有的 TRACE 指令将被忽略。这是为了避免交互调试意外中断。

? 和！可以视为开关，它们将使得 TRACE 指令分别进入不同的模式。

9.4.3 数字参数

在 TRACE 指令中，如果开启了交互式模式，可以使用正整数指定在调试时跳过的“暂停”次数；如果指定的是一个负数，那么接下来的指定数目的语句，都不会被跟踪，包括调试都将被禁用，指定数目之后的语句将会继续进入交互式调试模式。

```
TRACE ?I
A = 1
B = 2
C = 3
D = 4
IF (A > B) | (C < 2 * D) THEN
SAY 'At least one expression was true.'
ELSE
SAY 'Neither expression was true.'
/* 调试输出：*/
12 *-* A = 1
>L> "1"
+++ Interactive trace. TRACE OFF to end debug, ENTER to continue.
```

```
/* 此时输入 TRACE 3,后面的 3 句语句将被 REXX 跟踪执行,但不是交互式的,运行不会“暂停”,
运行情况如下;另外如果输入 TRACE -3,则后面的 3 句执行将根本不会被跟踪执行 */
13 *-* B = 2
>L> "2"
14 *-* C = 3
>L> "3"
15 *-* D = 4
>L> "4"
16 *-* IF (A > B) | (C < 2 * D)
>V> "1"
>V> "2"
⋮
/* 运行到第四个语句,REXX 再次进入交互式调试模式 */
*-* THEN
17 *-* SAY 'At least one expression was true.'
>L> "At least one expression was true."
At least one expression was true.
```

9.4.4 TRACE 指令输出格式

每条跟踪语句的输出根据它嵌套的深度来自动调整格式。编译器会先用？替换所有编码在数据中的控制字符，如果有结果输出，将空两个字符的位置并用双引号括起来，突出前后的空格。每行会有一个标号，如果超过了 99999，将被从左侧截断，并通过？来提示截断操作，如 100354 会显示为？00354。并且，每行跟踪结果的输出都有一个 3 位字符的前缀来标识跟踪的数据类型，这些前缀有如下几种类型。

（1）*-*标识来源于一条语句，即程序中真实的数据。

（2）+++标识一条跟踪信息，可能是非零的返回码、交互式调试时的提示信息、语法错误提示等。

（3）>>>标识一个表达式的解析结果或子例程的返回码。

（4）>.>标识在数据解析时赋给占位符的值。

```
TRACE 'R'
MyVar = 'This is some data'
PARSE VAR MyVar token1 . token2 .
EXIT

/* R 模式下,当有多个参数时,分别传递赋值*/
2 *-* MyVar = 'This is some data'
3 *-* PARSE VAR MyVar token1 . token3 .
   >>>   "This"
   >>>   "some"
4 *-* EXIT
```

表 9-1 列出一些输出前缀，这些前缀只有指定 TRACE I 时才会显示。

表 9-1　指定 TRACE I 才会显示的一些前缀

前缀	说　　明
>C>	标识一个复合变量在解析替换之后的名字
>F>	标识一个函数调用的结果
>L>	标识一个字符串，可以是未初始化的或字符常量
>O>	标识一个两项运算的结果
>P>	标识一个前缀运算的结果
>V>	标识一个变量的内容

如下例，使用 TRACE I 指令跟踪一段表达式的计算和判断。

```
x = 9
y = 2
TRACE I
IF x + 1 > 5 * y THEN
SAY 'x is big enough.'
ELSE NOP

/* 运行程序后，屏幕显示如下 */
9 *-* IF x + 1 > 5 * y      /* 显示被跟踪的行数为 9 以及对应的语句 */
>V> "9"                     /* 变量 x 的值为 9 */
>L> "1"                     /* 第二个输入的常量值为 1 */
>O> "10"                    /* x+1 的结果为 10 */
>L> "5"                     /* 下一个输入的常量值为 5 */
>V> "2"                     /* 变量 y 的值为 2 */
>O> "10"                    /* 5*y 的结果为 10 */
>O> "0"                     /* 最终运算 10>10 的结果为假（0）*/
```

如果在 TRACE 指令中未指定任何的参数，或一条表达式的解析结果为空，那么将使用默认的格式，即 TRACE N，! 和？都是关闭的。

9.5　中断程序的执行

在 TSO/E 环境下，可以使用 HI 或者 EXECUTIL HI 命令挂起当前执行的程序，或暂停对当前执行语句的翻译过程，常用于终止循环等。使用 HI 命令后，数据栈将被清空。同时，在 TSO/E 和非 TSO/E 环境下都可使用 IRXIC 例程来执行 HI 命令。

```
6 *-* D =4
   >>>   "4"
executil hi
6 +++  executil hi
```

```
6 +++ D =4
Error running TEST, line 6: Program interrupted
```

HE 命令也可以中断程序的执行。与 HI 不同，HE 命令立即执行而不是等到控制权返回到 REXX 程序后。如果 REXX 程序调用一个非 REXX 的外部例程，或者正在执行一个宿主命令，那么当使用 HI 命令时，程序会等到返回 REXX 环境下时再被挂起。如果调用的外部程序不能正常返回到原 REXX 程序，或宿主命令被挂起以致无法返回时，可以使用 HE 命令来中断当前程序的执行。EXECUTIL 和 IRXIC 命令不支持 HE 操作。

```
5 *-* C = 3
   >>>   "3"
|  /* 按下中断键(PA1) */
ENTER HI TO END, A NULL LINE TO CONTINUE, OR AN IMMEDIATE COMMAND+ -
HI

6 *-* D =4
  >>>   "4"
6 +++ D =4
 Error running TEST, line 6: Program interrupted
```

在程序中，可以用 TS 或 EXECUTIL TS 命令开启对程序的跟踪，用 TE 或 EXECUTIL TE 命令关闭跟踪，和 TRACE O 指令的作用相同，适用于在非交互式调试的模式下结束跟踪程序。

如果试图中断在 REXX 程序中执行的宿主命令，编译器会停止宿主命令的执行，并对 RC 赋值–1。这种情况下，编译器不会提示用户输入立即命令（Immediate Command），如 TS、HI 等，也就是不能对其进行跟踪和挂起的操作。如果试图中断在 ISPF 面板中执行的 REXX 程序，ISPF 会列出提示信息，表明当前的命令正被终止，并且关闭当前屏幕输出，同样不会有 TS 和 HI 等命令的操作。

9.6 交互式调试工具的使用

交互调试工具使用户能够交互地控制程序的执行。在交互过程当中，用户可以单步执行指令、插入指令、重新执行上一步指令等。使用交互调试工具，就像 Linux 下使用 GDB 一样，可以更为高效地调试 REXX 程序。事实上，交互调试可以这样理解：为非交互的 TRACE 添加断点，使得每一次跟踪调试后，程序暂停执行并等待用户输入，以此达到交互的目的。

启动交互调试工具有两种方法，一种是使用 TRACE 指令的?选项，另一种是使用 EXECUTIL TS 命令。两种方法的区别在于，如果 REXX 程序调用了其他程序或函数，则 TRACE ? 指令不能跟踪外部程序，但是可以在程序返回后继续跟踪，而 EXECUTIL TS 命令则可以连同外部程序一起调试跟踪。

9.6.1 TRACE ?命令

使用 TRACE ?指令可以开启交互式调试模式，并且会提示用户交互式调试已开启，可以在 TSO/E 环境下的当前控制台终端进行 REXX 程序的调试。其使用方法在第 2 章已经提及，这里再简单回顾一下。为了使用交互调试，用户需要在程序中加入 TRACE ? 和跟踪选项。例如 TRACE ?I（TRACE ?Intermediates）或 TRACE ?R（TRACE ?Result）。需要注意的是，?必须在选项之前，并且?和选项之间不能有空格。例如：

```
TRACE ?R
ARG dest dsname .
"TRANSMIT" dest "DA("dsname")"
IF RC = 0 THEN
SAY 'Transmit successful.'
ELSE
SAY 'Return code from transmit was' RC
```

对于上述示例，在调试过程当中，若传给程序的参数是 node1.mel 和 new.exec，则每一步的调试界面将类似于以下程序。

```
2 *-* ARG dest dsname.
>>> "NODE1.MEL"
>>> "NEW.EXEC"
>.> ""
+++ Interactive trace. TRACE OFF to end debug, ENTER to continue.

3 *-* "TRANSMIT" dest "DA("dsname")"
>>> "TRANSMIT NODE1.MEL DA(NEW.EXEC)"
0 message and 20 data records sent as 24 records to NODE1.MEL
Transmission occurred on 05/21/2010 at 11:40:01.

4 *-* IFRC = 0
>>> "1"
*-* THEN

5 *-* SAY 'Transmit successful.'
>>> "Transmit successful."
Transmit successful.
```

通常，在进入交互式模式后，其后所有的 TRACE 指令都将被忽略，并且程序会在所有被跟踪的指令后暂停，此时，用户可以执行以下 3 种操作。

（1）跳过当前步骤的调试。输入空行，不包括任何字符和空格，使编译器继续执行，直到下一次暂停。例如在 TRACE ?A 指令中，这种方法类似于单步调试，可以查看每一次跟踪暂停时的结果而不做任何操作。同理，若使用的是 TRACE ?E，则按回车键会跳转到下一条出错的命令处。

（2）重复执行上一条语句。输入等号（=），不包括空格，使编译器重新执行上一条被跟踪的语句，执行的语句指的是 REXX 程序里的指令，不包括用户在调试过程中插入的指令。这种方法适用于当用户发现程序中的一些变量赋值错误等，更正后查看是否改变了原来程序的执行，就可以重新执行一次，并且在执行之后，程序会再次暂停。

（3）其他执行指令。其他输入，可以是一行或多行 REXX 指令（包括调用其他 REXX 程序或 CLIST），并且立即执行这些输入的语句，也可以输入 TRACE 命令来改变调试选项。与 INTERPRET 指令的执行相同，输入的语句如果是成对的必须要匹配，如 DO-END。执行的语句如果有语法错误，将显示错误信息，并提示用户再次输入。在语句的执行期间，不会再进一步跟踪。如果用户想调试下一条语句，可输入 TRACE 指令停止当前步的调试。

一个常见的用法，是插入一条赋值语句，来改变一个变量的值，从而改变分支语句的执行，以此来覆盖各个分支。见下例，在调试时，可以输入 VAR = 0 来改变分支的执行。

```
IF VAR = 0 THEN
DO
INSTRUCTION1
END
ELSE
INSTRUCTION2
```

注意，有些指令反复执行，可能会产生一定的风险，因为在这些指令执行之后编译器不会暂停，即使它们处于被跟踪的范围之内。这些语句包括：

① 循环中第二次或之后执行的 DO 语句。

② END 语句。

③ THEN、ELSE、OTHERWISE 和空语句。

④ RETURN 和 EXIT 语句。

⑤ SIGNAL 和 CALL 语句。

⑥ 引发跟踪事件的语句。

⑦ 产生语法错误的语句，可以被 SIGNAL ON SYNTAX 跟踪，但是不能重复执行。

在进入交互调试模式后，可以使用如下方法退出。

① 输入 TRACE O（OFF）关闭调试跟踪。

② 输入 TRACE 而没有任何选项，关闭交互调试，置 TRACE 选项为默认值，保留并继续跟踪。

③ 再次输入 TRACE ?关闭交互调试，不把 TRACE 选项置为默认值，而是保留原值，并继续跟踪。

④ 再次输入 TRACE ?与新的选项，则继续跟踪，并使用新的调试选项。

⑤ 在调试时，输入 EXIT 指令结束调试，因为它可以终止程序的运行，所以调试也就随之终止了。

9.6.2 EXECUTIL TS 命令

EXECUTIL TS 命令和 TRACE ?R 指令的效果是一样的，只不过 EXECUTIL TS 还可以调试 REXX 程序调用的外部程序，而 TRACE ?R 则不能。

EXECUTIL TS 命令发起的位置可以是 REXX 程序内部、TSO READY 模式或 ISPF 面板。

1. 在 REXX 程序内部

如果在 REXX 程序内部使用 EXECUTIL TS 命令，则该程序的剩余部分以及它所调用的程序和同时正在运行的其他 REXX 程序都将被跟踪。这个跟踪过程将持续到所有被跟踪的程序执行结束或者使用任何一个程序中使用了 EXECUTIL TS 命令。如果一个 REXX 程序调用了一个 CLIST 程序，而该 CLIST 程序中使用了 EXECUTIL TS 命令，那么当程序返回到调用的 REXX 程序时，该 REXX 程序将被跟踪。

```
EXECUTIL TS
A=1;B=2;C=3;D=4;
IF (A>B) | (C< 2*D) THEN
SAY 'AT LEAST ONE IS CORRECT'
ELSE
SAY 'BOTH ARE WRONG'
EXIT

7 *-* IF (A>B) | (C< 2*D)
  >>>   "1"
  *-*  THEN
8 *-*  SAY 'AT LEAST'
   >>>    "AT LEAST ONE IS CORRECT"
AT LEAST ONE IS CORRECT
11 *-* EXIT
```

2. 在 TSO READY 模式下

在 READY 模式下，输入 EXECUTIL TS，下一个从 READY 模式启动的 REXX 程序将被调试跟踪。并且，该 REXX 程序调用的其他程序，也会被调试。输入 EXECUTIL TE 可以结束单步跟踪模式。

```
READY
EXECUTIL TS
```

3. 在 ISPF 面板下

因为在 READY 模式下可以调试，同样，在 ISPF COMMAND 面板也可以输入 EXECUTIL TS 命令，如图 9-1 所示，并且在 EXECUTIL TS 命令后的下一个 REXX 程序将被调试，而它所调用的程序也会被跟踪。注意，如果使用分屏，在两个 ISPF 面板中的程序是互不影响的，即对一个面板中的程序进行跟踪，不会影响到另一个面板中的程序

的执行，因为它们是隶属于两个不同的环境的。

```
  Menu  List  Mode  Functions  Utilities  Help
-------------------------------------------------------------------------------
                              ISPF Command Shell
 Enter TSO or Workstation commands below:

 ===> EXECUTIL TS

 Place cursor on choice and press enter to Retrieve command

 => EXECUTIL TE
 => TEST
 => EXECUTIL TS
 =>
 =>
```

图 9-1　输入 EXECUTIL TS 命令后的界面

```
+++ "TSO COMMAND TEST SYS00039 USERID.REXX ? TSO ISPF ?"
     1 *-*                          /***REXX***/
     2 *-* IF (A>B) | (C< 2*D)  /* 进入交互式调试模式,和 TRACE ?R 相同 */
```

可以使用 EXECUTIL TE 命令结束调试,既可以在交互式调试中执行也可以在 REXX 程序中事先定义好。EXECUTIL TE 命令可以终止当前程序的跟踪，以及它所调用的程序和其他正在运行的程序。

```
+++ Interactive trace.  TRACE OFF to end debug, ENTER to continue. +++
3 *-* A =1
  >>>   "1"
EXECUTIL TE                      /* 在屏幕上输入 EXECUTIL TE 结束跟踪 */
 AT LEAST ONE IS CORRECT         /* 调试跟踪结束,返回 */
```

注意 TE 命令不能在 READY 模式下输入，因为该环境下没有正在运行的 REXX 程序。但是，如果在 READY 模式下指定了 EXECUTIL TS 命令准备对下一个程序进行跟踪，而后又指定了 EXECUTIL TE 命令，那么将取消对下一个程序的跟踪。

9.7　IRXIC 例程

TSO/E 还提供了一些可用的外部例程来支持和编译器的交互操作。其中有一个例程是 IRXIC，通过它可以调用 HI、HT、RT、TS 和 TE 命令。该例程可以用在汇编或其他高级语言写成的程序中，来控制 REXX 程序的执行和跟踪过程。

想要运行 TSO/E 提供的这些外部例程，需要有一个编译器环境，即可以运行 REXX 程序的环境，在 TSO/E 环境下，通过登录过程可以创建一个运行环境。关于 TSO/E 外部例程的使用可参见 TSO REXX Reference 红皮书。

程序可以通过 CALL 或 LINK 宏来调用 IRXIC 例程，指定 IRXIC 为入口名称。首先

程序需要创建一个参数列表，并且将其地址放入 1 号寄存器中。在参数列表中，指定要执行的操作等信息，具体如下。

（1）参数 1（4 位），指定要使用的命令，可以是 HI、HT、RT、TS 和 TE。HI 表明暂停对 REXX 程序的翻译过程；HT 表明停止 REXX 程序的输出，如 SAY 指令等；RT 表明恢复 REXX 程序的输出；TS 和 TE 分别是开启和停止对程序的跟踪。

（2）参数 2（4 位），指定参数 1 所指向的命令的长度。

（3）参数 3（4 位），指定要执行的命令所处的环境，该参数是可选的，如果指定了，该地址必须是有效的，因为 IRXIC 不会对该地址进行检查；如果未指定，则使用 0 号寄存器中保存的地址值。

（4）参数 4（4 位），保存返回码，该参数是可选的，如果未指定就保存在 15 号寄存器中，如果指定了，就在该参数和 15 号寄存器中都保存返回码。

IRXIC 例程的返回码如表 9-2 所示。

表 9-2 IRXIC 例程的返回码

返回码	描 述
0	执行成功
20	执行有错误，通过系统信息给用户提示
28	IRXIC 例程无法找到程序的编译执行环境
32	参数列表包含无效信息，或者有未指定的必需参数



REXX 综合案例

本章结合前面的内容，根据主机系统管理与维护的日常工作需求，提供给用户几个典型的 REXX 应用。

10.1 综合案例一

1. 案例描述

读取 XML 数据，根据 XML 内容来动态创建与修改文件。

2. 业务逻辑

REXX 程序从一个 XML 文件（某个顺序数据集）中动态读取数据集的信息，根据这些信息来创建数据集。案例中设计了 3 个数据集：顺序数据集（FB/80）、扩展分区数据集（FB/80）和可以容纳执行模块（Load Module）的扩展分区数据集（U/0）。

数据集创建完之后，要求在顺序数据集中新增一条记录，在扩展分区数据集中增加一个成员，成员有一条记录内容，最后在该扩展分区数据集成员内“追加”（不覆盖原内容）一条新记录。

3. 技术点

通过 TSO 命令 ALLOCATE 命令完成数据集的分配创建；通过 EXECIO 提供的 I/O 操作完成对各种数据集的读写。通过数据栈的 QUEUE 以及 STACK 操作完成数据追加。

4. 案例代码

```
/***********       REXX Sample Code 1                  ************/
/*****************************************************************/
/*              Read dsnames from XML file                       */
/*    SAMPLE.DSNAMES holds XML records as following :            */
/*             <dsn1>SAMPLE.DS1.SEQ</dsn1>                       */
/*             <dsn2>SAMPLE.DS2.PDSE</dsn2>                      */
/*             <dsn1>SAMPLE.DS3.PDS</dsn3>                       */
/*****************************************************************/
        Address TSO
"ALLOC DA('SAMPLE.DSNAMES') F(DSNAMES) OLD REUSE"
```

```
"EXECIO * DISKR DSNAMES  (stem dsn. finis"
"FREE F(DSNAMES)"

/*************** parse dsn into three data set names *************/
parse upper value dsn.1 with "<DSN1>" seq_name "</DSN1>"
parse upper value dsn.2 with "<DSN2>" pdse_name "</DSN2>"
parse upper value dsn.3 with "<DSN3>" pds_name "</DSN3>"

/*************** Allocate a (FB/80) SDS  *************************/
"ALLOCATE DATASET('"seq_name"') F(DAT1) NEW SPACE(50,20)",
"DSORG(PS) RECFM(F,B) LRECL(80) BLKSIZE(8000)"
"FREE F(DAT1)"
say "A SDS has been allocated with the name " seq_name

/*************** Allocate a (FB/80) PDSE ************************/
"ALLOCATE DATASET('"pdse_name"') F(DAT2) NEW SPACE(50,20) DIR(10)",
"DSORG(PO) RECFM(F,B) LRECL(80) BLKSIZE(8000) DSNTYPE(LIBRARY)"
"FREE F(DAT2)"
say "A PDSE has been allocated with the name " pdse_name

/*************** Allocate a (U/0) PDS ****************************/
"ALLOCATE DATASET('"pds_name"') F(DAT3) NEW SPACE(50,20) DIR(10)",
 "DSORG(PO) RECFM(U) LRECL(0) BLKSIZE(8000) DSNTYPE(PDS)"
 "FREE F(DAT3)"
say "A PDS has been allocated with the name " pds_name

/***************  Write a line into new SDS  ********************/
"ALLOC DA('"seq_name"') F(SEQDD)  OLD REUSE"
QUEUE "New line   "
"EXECIO * DISKW  SEQDD (FINIS "
"FREE F(SEQDD)"

/*********Create a new member with 1 record to PDSE ***************/
"ALLOC DA('"pdse_name"(MEMBER1)') F(PDSEDD)  OLD REUSE"
/*     Write a line into new member  */
QUEUE "Record 1   "
"EXECIO * DISKW  PDSEDD (FINIS "
"FREE F(PDSEDD)"

/*********Append a new record into the member without overriding it***/
"ALLOC DA("pdse_name"(MEMBER1)) F(PDSEDD) SHR REUSE"
"NEWSTACK"          /* Create a new data stack for input only */
"EXECIO * DISKR PDSEDD (FINIS"
QUEUE 'New Record 2'  /* Append a new record into stack  */
QUEUE '' /* Add a null line to indicate the end of the information */
"EXECIO * DISKW PDSEDD (FINIS"
```

```
"DELSTACK"          /* Delete the new data stack */
"FREE F(PDSEDD)"
say "The content of the member in PDSE has been updated "
```

10.2　综合案例二

1. 案例描述

RACF 作业的自动提交执行。

2. 业务逻辑

通过 REXX 程序自动提交 RACF 作业，设计了以下 3 个单独的作业：

（1）创建一个组，若成功，则继续执行下一作业；否则给出作业错误消息。

（2）在该组中创建一个用户，若成功，则继续执行下一作业；否则给出作业错误消息。

（3）保护该用户数据集，在上面执行过程中如果失败，则终止程序给出具体错误消息。

3. 技术点

通过 QUEUE 构建需要执行的 JCL 作业，通过 SUBMIT 命令进行提交；使用 OUTTRAP 截取 SUBMIT 提交后的终端输出，并由此获取 JOBNAME 以及 JOBID。

用 QUEUE 构建 SDSF 的批处理命令，并将其写入 SDSF 的批处理标准输入接口文件 isfin 并执行，将 RACF 作业中的标准输出 SYSTSPRT 复制到临时文件处理。判断临时文件内容的第三行消息确定该作业是否顺利执行（出现 READY 为成功，否则为具体的 RACF 错误消息）。

4. 案例代码

```
/***********           REXX Sample Code 2           ************/
/***************************************************************/
/*             build RACF JCLs using QUEUE                     */
/*   subroutine CHECK_MSG sets maxcc to determine the result   */
/*            step1 : add a group GRPA1                        */
/*            step2 : create a user ID001                      */
/*            step3 : protect the dataset for ID0001           */
/***************************************************************/
maxcc = 0
/***************    step 1 : add a new group  ******************/
newstack
QUEUE "//ADDGRP  JOB NOTIFY=&SYSUID "
QUEUE "//IBMUSER  EXEC PGM=IKJEFT01,DYNAMNBR=20,REGION=512K "
QUEUE "//SYSPRINT DD  DUMMY "
```

```
QUEUE "//SYSTSPRT DD  SYSOUT=* "
QUEUE "//SYSTSIN  DD  * "
QUEUE "     AG GRPA1 SUPGROUP(GRPA) OWNER(GRPA)"
QUEUE "//                                   "
QUEUE "$$                                   "
O = OUTTRAP("OUTPUT.",,"CONCAT")      /* Trap output of submitting */
ADDRESS TSO "SUBMIT * END($$)"
O= OUTTRAP(OFF)
delstack
SAY OUTPUT.2
PARSE VAR OUTPUT.2 'JOB' jobname'('jobid')'STATUS
SAY "The job submitted is " jobname
SAY "The jobid assigned is " jobid
CALL CHECK_MSG
if maxcc<>0 then
   exit
else
   nop

/**************step 2: create a new user**************/
newstack
QUEUE "//ADDUSR  JOB NOTIFY=&SYSUID "
QUEUE "//IBMUSER  EXEC PGM=IKJEFT01,DYNAMNBR=20,REGION=512K "
QUEUE "//SYSPRINT DD  DUMMY "
QUEUE "//SYSTSPRT DD  SYSOUT=* "
QUEUE "//SYSTSIN  DD  * "
QUEUE "     AU ID001 DFLTGRP(GRPA1) PASS(PASS)"
QUEUE "//                                   "
QUEUE "$$                                   "
O = OUTTRAP("OUTPUT.",,"CONCAT")
ADDRESS TSO "SUBMIT * END($$)"
O= OUTTRAP(OFF)
delstack
SAY OUTPUT.2
PARSE VAR OUTPUT.2 'JOB' jobname'('jobid')'STATUS
SAY "The job submitted is " jobname
SAY "The jobid assigned is " jobid
CALL CHECK_MSG
if maxcc<>0 then
   exit
else
   nop

/**************step 3: protect resource**************/
newstack
QUEUE "//PROTSD  JOB NOTIFY=&SYSUID "
```

```
QUEUE "//IBMUSER  EXEC PGM=IKJEFT01,DYNAMNBR=20,REGION=512K "
QUEUE "//SYSPRINT DD  DUMMY "
QUEUE "//SYSTSPRT DD  SYSOUT=* "
QUEUE "//SYSTSIN  DD  * "
QUEUE "     ADDSD 'ID001.**' uacc(none)       "
QUEUE "//                                   "
QUEUE "$$                                   "
O = OUTTRAP("OUTPUT.",,"CONCAT")
ADDRESS TSO "SUBMIT * END($$)"
O= OUTTRAP(OFF)
delstack
SAY OUTPUT.2
PARSE VAR OUTPUT.2 'JOB' jobname'('jobid')'STATUS
SAY "The job submitted is " jobname
SAY "The jobid assigned is " jobid
CALL CHECK_MSG
if maxcc<>0 then
   exit
else
   nop

/***********subroutine: check_msg*********************/
CHECK_MSG:
if(jobname = "") then do
return "Jobname empty"
end
if(jobid = "")  then do
return "JobId empty"
end
/* write SDSF Commands to QUEUE */
queue "SET CONFIRM OFF"
queue "OWNER *"
queue "PREFIX *"
queue "ST"
queue "SELECT "jobname jobid   /* locate current job  */
queue "AFD REFRESH"
queue "++?"
queue "FIND SYSTSPRT"          /* locate SYSTSPRT  */
queue "++S"
queue "PRINT FILE TEMP"        /* save output into temp file   */
queue "PRINT"
queue "PRINT CLOSE"
queue "END"
queue ""
"alloc dd(isfin)  new reuse",
     "recfm(f,b) lrecl(80) blksize(27920) space(1) tracks"
```

```
"alloc dd(isfout) new reuse ",
     "recfm(f,b,a) lrecl(301) blksize(27993) space(1) cyl"
 "ALLOCATE DATASET('SAMPLE.SDSF.TEMP') F(temp) NEW SPACE(50,20)",
 "DSORG(PS) RECFM(V) BLKSIZE(8000)"
 /* write input-dataset from QUEUE */
 "execio * diskw isfin (finis"
 /* SDSF-Call */
 ADDRESS ISPEXEC "SELECT PGM(ISFAFD) PARM('++25,80')" /*INVOKE SDSF */
 "free F(isfin,isfout,temp)"
 "ALLOC F(INPDD1) DA('SAMPLE.SDSF.TEMP') SHR"
 "execio * diskr INPDD1 (stem out. finis"
 "delete 'SAMPLE.SDSF.TEMP'"
/*  check the message in the 3 line of content */
 IF out.3 = 'READY'then
     do
       SAY jobname " has been run successfully"
       maxcc=0
     end
 else
     do
       say jobname " failed to complete successfully due to " out.3
       maxcc=12
     end
 return
```

10.3 综合案例三

1. 案例描述

控制台命令交互。

2. 业务逻辑

在 REXX 程序中，使用 MVS Console 命令启动一个 CICS，根据返回信息判断控制台命令是否执行成功；同时检测启动过程中是否需要用户命令应答，并输入相关消息。

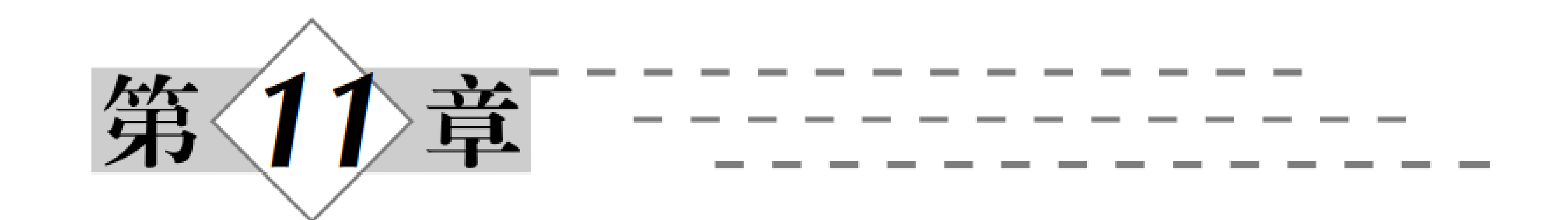
3. 技术点

通过 CONSPROF SOLDISPLAY(no) UNSOLDISPLAY(no)将输出信息暂存在消息缓存表中，并通过 CART()标定执行命令对应的输出信息，并用 GETMSG()取回暂存信息进行处理。

4. 案例代码

```
/***********       REXX Sample Code 3                ********** **/
/*****************************************************************/
/*            CONSOLE ACTIVATE to issue SYSCMD                   */
```

```
/*  use CART to identify the message associated with SYSCMD          */
/*  use DFHSI1517 as indicator of success message in CICS start-up */
/*  RC of GETMSG() :                                                 */
/*       0 : Processing successful; a message has been returned     */
/*       4 : Processing successful; a message not found             */
/*       8 : Processing successful; Attention Key to end GETMSG     */
/*     other: Processing unsuccessfully                              */
/*********************************************************************/
"CONSPROF SOLDISPLAY(no) SOLNUM(400) UNSOLDISPLAY(no) UNSOLNUM(500)"
"CONSOLE ACTIVATE"                            /* Open Console session */
"CONSOLE SYSCMD($S CICSAOR1) CART('DFHMSG')"
 rc= GETMSG(msg.,'sol','DFHMSG',,60) /* Retrieve Command Response  */
 if rc= 4 then
  do
     x= GETMSG(msg.,'sol','DFHMSG')
  end
/* DFHSI1517 as success message indicator */
if rc = 0 then
   do  i = 1 to msg.0
      if pos('DFHSI1517',msg.i)<>0 then
        do
         say "CICS has been started successfully "
         exit
        end
      else
        nop
   end
/* Issue Display System Request Command */
   "CONSOLE SYSCMD(D R,JOB=CICSAOR1) CART('RESP')"
/*  Retrieve Outstanding Msg */
    request = GETMSG(resp.,'EITHER','RESP',,60)
    if request = 0 then
       do
         say " Please reply all the outstanding messages "
         do ix = 1 to resp.0
            say resp.ix
         end
       end
    else
        say "GETMSG error retrieving message.  Return code is" rc
"CONSOLE DEACTIVATE"                     /* Close Console Session  */
```



REXX 实验

11.1 预备实验

1. 实验目的

登录 TSO。实验后，学生应该掌握 TSO 的登录。

2. 实验介绍

通过本实验，掌握登录 TSO 的步骤。

3. 实验步骤

（1）输入 LOGON　REXXUxx，登录 TSO，出现的界面如图 11-1 所示。

```
z/OS V1R5 Level  0403                                     IP Address  =
220.152.168.239
                                                   VTAM Terminal      = A06TSO28

                        Application Developer System

                                 //     OOOOOOO   SSSSSSS
                                //     OO     OO  SS
                        zzzzzz  //    OO     OO  SS
                          zz   //   OO     OO    SSSS
                        zz    //   OO     OO       SS
                      zz     //   OO     OO        SS
                    zzzzzz  //    OOOOOOO   SSSSSSS


                    System Customization    - ADCD.ZOSV1R5.*



 ===> Enter "LOGON" followed by the TSO userid  . Example  "LOGON IBMUSER " or
 ===> Enter L followed by the APPLID
 ===> Examples : "L TSO", "L CICS", "L IMS3270

LOGON REXXU01
```

图 11-1　用户登录 TSO

（2）进入 TSO 登录界面，输入初始密码，然后根据系统提示修改密码，如图 11-2 所示。

```
------------------------------- TSO/E LOGON -----------------------------------
IKJ56714A Enter current password for RACFU01

   Enter LOGON parameters below:                    RACF LOGON parameters:

   Userid    ===> REXXU01

   Password  ===>                                   New Password ===>

   Procedure ===> IKJ#REXX                          Group Ident  ===>

   Acct Nmbr ===> ACT#REXX

   Size      ===> 4096

   Perform   ===>

   Command   ===> ISPF

   Enter an 'S' before each option desired below:
           -Nomail         -Nonotice         -Reconnect         -OIDcard

PF1/PF13 ==> Help    PF3/PF15 ==> Logoff    PA1 ==> Attention    PA2 ==> Reshow
You may request specific help information by entering a '?' in any entry field
```

图 11-2　TSO 登录界面

（3）进入 TSO 界面，遇到***直接按回车键（右 Control 键）即可进入 ISPF 操作界面，如图 11-3 所示。

```
ICH70001I REXXU01  LAST ACCESS AT 15:56:23 ON SATURDAY, JUNE 2, 2007
IKJ56455I REXXU01 LOGON IN PROGRESS AT 18:08:05 ON JUNE 2, 2007
IKJ56951I NO BROADCAST MESSAGES
*****************************************************************
*                                                               *
*                     WELCOME STUDENT TO                        *
*                                                               *
*                                                               *
*                                                               *
*                   MAINFRAME  TECHNOLOGY TRAINING              *
*                                                               *
*                                                               *
*                                                               *
*                                                               *
*                                                               *
*                                                               *
*                                                               *
*                                                               *
*****************************************************************

***
```

图 11-3　TSO 登录过程

（4）ISPF 主界面如图 11-4 所示。

```
 Menu  Utilities  Compilers  Options  Status  Help
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
                           ISPF Primary Option Menu

0  Settings      Terminal and user parameters          User ID . : REXXU01
1  View          Display source data or listings       Time. . . : 18:10
2  Edit          Create or change source data          Terminal. : 3278
3  Utilities     Perform utility functions             Screen. . : 1
4  Foreground    Interactive language processing       Language. : ENGLISH
5  Batch         Submit job for language processing   Appl ID . : ISR
6  Command       Enter TSO or Workstation commands     TSO logon : ISPFPROC
7  Dialog Test   Perform dialog testing                TSO prefix: REXXU01
9  IBM Products  IBM program development products      System ID : P390
10 SCLM          SW Configuration Library Manager      MVS acct. : ACT#REXX
11 Workplace     ISPF Object/Action Workplace          Release . : ISPF 5.5
M  More          Additional IBM Products
MQ MQAPPs        MQSeries ISPF Utilities

     Enter X to Terminate using log/list defaults



Option ===>
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

图 11-4　ISPF 主界面

11.2　REXX 基础实验一

1. 实验目的

使用 REXXTRY 和 SAY 指令；用 REXX 执行算术运算和 I/O 操作。实验后，学生应该掌握以下技能。

- 使用 REXXTRY。
- 使用 SAY 指令。
- 进行语句续行。
- 读写终端。
- 用 REXX 进行算术运算。

2. 实验介绍

通过使用 REXXTRY 程序，了解各类 REXX 语句的含义和特点。

3. 实验步骤

（1）登录 TSO，输入以下 REXX 程序代码，并命名为 REXXTRY。

```
File Edit Edit _Settings Menu Utilities Compilers Test Help
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
EDIT       yourid.REXX.LAB(REXXTRY) - 01.02        Columns 00001 00072
```

```
Command ===>                                                Scroll ===> CSR
****** ************************* Top of Data ****************************
000100 /* INTERACTIVE REXX INSTRUCTION EXECUTION */
000200 SAY "EXEC REXXTRY ALLOWS YOU TO INTERACTIVELY EXECUTE "
000210 SAY "REXX INSTRUCTIONS . EACH INSTRUCTION STRING IS "
000220 SAY "EXECUTED WHEN YOU PRESS ENTER ."
000230 SAY "TO END , TYPE EXIT ."
000300 DO  FOREVER
000400     SAY "REXXTRY :"
000500     PARSE EXTERNAL LINE
000600     IF LINE = "EXIT " THEN LEAVE
000700     INTERPRET LINE
000800 END
****** *********************** Bottom of Data **************************
```

（2）在 ISPF 的命令行，输入 TSO EXEC 'yourid.REXX.LAB(REXXTRY)'，观察屏幕提示的内容。

（3）当运行 REXXTRY 的时候，输入以下语句，观察输出结果。

```
Say REXX beats java
Say Rexx 'beats' java
Say Rexx     'beats'    java
a = Rexx
b = eats
c = java
Say a b c
Say a b b c
Say a 'b' b c
Say a 'b' || b c
Say a 'b'b c
Say a b c
a = "Rexx"; b = "eats"; c = "Java beans for breakfast"
Say a b c
Say a || b || c
Say a || b || "fresh" c
Say 5 + 9 * 33 - 2 / (7 + (5*45))
Say 'c1'x
Say 'c2'x
Say '1100 0001'b
Say '1100 0011'b
Trace R
Say hello
Trace N
11111
Exit
```

（4）算术运算练习。要求编写一个 REXX 程序 MATH1，完成以下操作：

① 提示用户输入 2 个数字。

② 从键盘接收这 2 个数字。

③ 首先显示用户输入的数字内容，然后对它们进行加、减、乘和除法运算，并将结果显示在屏幕上。

例如，如果用户输入 25 和 4 这 2 个数字，REXX 程序的输出结果如下。

```
Please enter two numbers
25 4
You entered 25 and 4.
25 + 4 = 29
25 - 4 = 21
25 * 4 = 100
25 / 4 = 6.25
25 divided by 4 is 6 with a remainder of 1
```

11.3 REXX 基础实验二

1. 实验目的

进一步地了解 REXX 各指令的含义，学习条件分支语句的使用。实验后，学生应该掌握：

- 使用 Parse Arg 指令。
- 使用条件分支 IF-THEN-ELSE 语句。

2. 实验介绍

运用 REXXTRY 进行逻辑运算、条件分支 If-Then-Else 语句以及 Parse Arg 指令的实验；并对实验一的步骤（4）进行改进。

3. 实验步骤

（1）登录 IPSF，选择选项 6，进入 TSO 命令面板如图 11-5 所示。

```
  Menu  List  Mode  Functions  Utilities  Help
--------------------------------------------------------------------------------
                              ISPF Command Shell           3 Members processed
Enter TSO or Workstation commands below:

===>


Place cursor on choice and press enter to Retrieve command

=>
=>
```

图 11-5 TSO 命令面板

（2）请在系统管理员的帮助下，把 REXXTRY 程序放入到系统 REXX 库中。然后在 TSO 命令面板中执行如下命令。

```
1    REXXTRY chives dill fennel sage thyme
2    Say 555 = 555
3    Say 555 = 556
4    Say 555 = 555.0000000001
5    a = 1; b = 2; c = 3
6    Say a b c
7    Say a = a
8    Say a = b
9    Say (a + b) = c
10   answer = " YES "
11   Say answer = "YES"
12   Say answer == "YES"
13   Say answer = "yes"
14   If answer = 'YES' Then Say 'The answer is yes'; Else Say 'The answer is no'
15   Arg parms
16   Say parms
17   Parse Arg parms
18   Say parms
19   Arg a b c d e f
20   Say a
21   Say b
22   Say c
23   Say d
24   Say e
25   Say f
26   Arg fulllist
27   Parse Var fulllist item fulllist
28   Say item
29   Say fulllist
Repeat steps 27 to 29 until you understand what is happening here.
```

（3）对实验一（步骤（4））的 MATH1 进行如下改变，生成 REXX 新程序 MATH2，要求：

① 在命令行中直接输入算术运算所需的 2 个数字参数。

② 如果第 2 个数字为 0，则不要进行除法运算。

例如，如果用户在命令行中键入数字 21 和 0，则 REXX 程序的输出结果如下。

```
COMMAND ===> tso math2 21 0
You entered 21 and 0.
21 + 0 = 21
21 - 0 = 21
```

```
21 * 0 = 0
The second number is 0: division cannot be done.
```

11.4 REXX 基础实验三

1. 实验目的

学习在 REXX 程序中运用循环控制语句。实验后，学生应该掌握使用循环控制语句。

2. 实验介绍

对实验二中的 MATH2 程序进行改进，加入循环控制语句。

3. 实验步骤

对实验二（步骤（3））的 MATH2 进行如下改进，生成 REXX 新程序 MATH3，要求如下：

① 必须接收到 2 个参数，否则程序将一直提醒用户输入。

② 用户输入 Exit 可退出运算。

例如，MATH3 程序的一个有效示例如下：

```
COMMAND ===> tso math3
Please enter two numbers:
10
Please enter two numbers:
10 5
You entered 10 and 5.
10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
10 / 5 = 2
10 divided by 5 is 2 with a remainder of 0
```

11.5 REXX 函数与子例程调用

1. 实验目的

通过实验，学习 REXX 中函数和子例程的调用方式。实验后，学生应该掌握编写和调用 REXX 函数以及子例程。

2. 实验介绍

分别用函数和子例程两种方式实现求阶乘 $n!$，体会函数和子例程编写以及调用方式的不同；子函数和子例程分别用递归和非递归两种逻辑实现。

3. 实验步骤

编写 REXX 程序。要求：

① 函数用递归逻辑完成阶乘运算。

② 子例程用非递归逻辑完成阶乘运算。

③ 在主调程序中分别调用函数和子例程计算不同数值 n 的阶乘运算结果 $n!$，并相互比较运算结果。

有效示例如下：

```
COMMAND ==>TSO Lab 5
Please enter the number :
6
THE FACTORIAL OF 1 FROM FUNCTION   IS : 1
THE FACTORIAL OF 1 FROM SUBROUTINE IS : 1
THE FACTORIAL OF 2 FROM FUNCTION   IS : 2
THE FACTORIAL OF 2 FROM SUBROUTINE IS : 2
THE FACTORIAL OF 3 FROM FUNCTION   IS : 6
THE FACTORIAL OF 3 FROM SUBROUTINE IS : 6
THE FACTORIAL OF 4 FROM FUNCTION   IS : 24
THE FACTORIAL OF 4 FROM SUBROUTINE IS : 24
THE FACTORIAL OF 5 FROM FUNCTION   IS : 120
THE FACTORIAL OF 5 FROM SUBROUTINE IS : 120
THE FACTORIAL OF 6 FROM FUNCTION   IS : 720
THE FACTORIAL OF 6 FROM SUBROUTINE IS : 720
```

11.6　数据解析实验

1. 实验目的

学习如何在 REXX 中进行数据解析（Parsing）。实验后，学生应该掌握使用 Parse 的有关指令对数据进行解析。

2. 实验介绍

使用 REXXTRY 对数据进行分析。

3. 实验步骤

执行 REXXTRY 程序，然后输入以下命令，观察系统输出结果。

```
quote = 'Experience is the best teacher.'
parse var quote word1 word2 word3
say word1
say word2
say word3
parse var quote word1 word2 word3 word4 word5 word6
say word1
say word2
say word3
```

```
say word4
say word5
say word6
parse value quote with word1 word2 word3
say word1
say word2
say word3
parse var quote word1 word2 15 word3 3 word4
say word1
say word2
say word3
say word4
parse var quote 15 v1 +16 =12 v2 +2 1 v3 +10
say v1
say v2
say v3
parse var quote 1 v1 +11 v2 +6 v3 -4 v4
say v1
say v2
say v3
say v4
first = 7
parse var quote 1 v1 =(first) v2 +6 v3
say v1
say v2
say v3
```

11.7 REXX 错误处理与调试机制实验

1. 实验目的

掌握 REXX 的错误处理方法以及常用的调试机制。实验后，学生应该掌握：

- 使用基本的错误处理指令。
- 基本的程序调试。

2. 实验介绍

使用异常与错误处理指令 SIGNAL 以及 TRACE。

3. 实验步骤

（1）在前面的所有实验程序中加入以下两条容错指令，并在相应的指令下书写相应的恢复代码，然后重新执行 REXX 程序，观察容错处理效果。

```
SIGNAL ON NOVALUE
SIGNAL ON ERROR
```

执行 SIGNAL ON ERROR 后的界面如图 11-6 所示。

```
000001 /* REXX */
000002 SIGNAL ON ERROR
000003 "ALLOC DA(new.data) LIKE(old.data)"
⋮
000008 "LISTDS ?"
  ⋮
000011 EXIT
000012
000013 ERROR:
000014 SAY 'The return code from the command on line' SIGL 'is' RC
000015 /* 以上将显示：The return code from the command on line 8 is 12 */
```

图 11-6　SIGNAL ON ERROR 示例

（2）为所有完成的程序添加交互调试命令 TRACE ?R，观察程序执行。

11.8　执行宿主命令实验

1. 实验目的

掌握如何执行宿主命令（Host Command）。实验后，学生应该掌握宿主命令的调用。

2. 实验介绍

本实验步骤（1）将使用 REXXTRY 调用各类不同的命令。

3. 实验步骤

（1）执行 REXXTRY 程序，然后输入以下命令，观察系统输出结果。

1　"avg 5, 7"

2　say rc

3　address ispexec

4　say rc

5　"avg 5, 7"

6　address ispexec

7　"control errors return"

8　say rc

9　"math2 6 9"

10　say rc

11　"display panel(XXXXXXXX)"

这里 XXXXXXXX 可以是下面任何一个面板名称：DGTHDU50、ERB649I0、ICQDL10H、FLMHTSOT、ISPOPT1、ISR91192

12　say rc

输入以上列表中其他的面板名称，观察返回码；输入一个不存在的面板名称 NONEXIST. 观

```
  察输入码.
address link
"iefbr14"
say rc
address tso 'ALLOC DA(NEW.DATA) LIKE(OLD.DATA) NEW'
```

其中 NEW.DATA 和 OLD.DATA 分别为对应的数据集名称。

（2）编写一个程序，询问用户是否了解功能键（即 PF 键）的作用，如果用户回答为 N 或者 No，则转换当前宿主环境为 ISPEXEC 并调用 ISPF 服务命令显示帮助面板 ISPOPT3C。

11.9　REXX 构建并提交 JCL 作业

1. 实验目的

掌握用 REXX 构建并提交 JCL 的方法。实验后，学生应该掌握使用 REXX 语句构建 JCL 并提交一个 JCL 作业。

2. 实验介绍

构建一个使用 IKJEFT01 的 JCL 作业，并在后台用批处理方式调用前面完成 REXX 程序的 Math2 程序。

3. 实验步骤

（1）编写一个 REXX 程序，构建下面所示的 JCL，程序片段如下，其中 userid 为 TSO 用户 ID。

```
//useridA JOB 'ACCOUNT, DEPT, BLDG','PROGRAMMER NAME',
// CLASS=A, MSGCLASS=A, MSGLEVEL= (1,1)
//REXX      EXEC PGM=IKJEFT01, DYNAMNBR=30, REGION=4096K
//SYSEXEC    DD   DSN=yourid.REXX.LAB, DISP=SHR
//SYSTSPRT   DD   SYSOUT=A
//SYSTSIN    DD   *
%math2 25 5
/*
//
```

（2）用该 JCL 在 REXX 基础实验二中的 Mah2 程序，并传入 25、5 两个参数。

（3）在 SDSF 中查看 Math2 的运行结果。

11.10　REXX 调用 ISPF 服务实验

1. 实验目的

掌握调用 ISPF 服务的基本方法。实验后，学生应该掌握调用 ISPF 服务。

2. 实验介绍

调用 ISPF 的相关服务，完成已存在数据集之间的成员复制。

3. 实验步骤

编写 REXX 程序 COPYMEM，完成以下操作：

① 输入复制的源数据集名称以及成员名称。

② 输入复制的目标数据集名称以及目标成员名称（该目标数据集为已经创建存在的分区数据集）。

③ 源数据集成员被复制到目标数据集下，并命名为新的指定名称。

有效示例如下：

```
ENTER THE SOURCE DATASET AND ITS MEMBER TO COPY :
yourid.SOURCE.LIB MEM1
ENTER THE DESTINATION DATASET AND ITS MEMBER :
yourid.TARGART.LIB MEM2
COPYING MEMBER COMPLETED !
```

11.11　ISPF 编辑宏（Edit Macro）实验

1. 实验目的

学习使用 ISPF 编辑宏（Edit Macro）。实验后，学生应该掌握 ISPF 编辑宏的编写和使用。

2. 实验介绍

本实验利用 ISPF 编辑宏实现为文本添加头信息并删除多余行。

3. 实验步骤

用 REXX 编写编辑宏 REEDIT，实现以下功能：

① 将分区数据集成员的源文本中所有第一列标为*的行删除。

② 在分区数据集的成员文本第一行添加一个头行注释，例如/* Automatically Generate Head*/。

③ 在分区数据集的成员文本的第二行添加相应的程序员信息和当前日期信息。

注意：在运行编辑宏 REEDIT 之前，需要进入 TSO 命令面板，完成下面两个步骤：

- 使用 ALLOCATE FI(SYSUEXEC) DA('***yourid***.REXX.LAB') SHR REU 命令为 REEDIT 所在的分区数据集分配对应的 DD 名（如果使用 SYSPROC 或 SYSEXEC 会涉及权限和管理问题，推荐使用 SYSUEXEC 以及 SYSUPROC）。
- 使用 ALTLIB ACTIVATE USER(EXEC)命令使当前的编辑宏 REEDIT 可用。

REEDIT 程序源文本示例如图 11-7 所示。

输入 REEDIT 命令并按回车键，运行效果如图 11-8 所示。

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       TE02.REXX.LAB(TEXT) - 01.00                    Columns 00001 00072
****** ***************************** Top of Data ******************************
000001 *TEST1: THIS LINE WILL BE DLETED BY EDIT MACRO
000002 TEST2
000003 TEST3
000004 *TEST4: THIS LINE WILL BE DLETED BY EDIT MACRO
****** **************************** Bottom of Data ****************************


Command ===> REEDIT                                           Scroll ===> CSR
```

图 11-7　数据集编辑界面

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       TE02.REXX.LAB(TEXT) - 01.00                    Columns 00001 00072
****** ***************************** Top of Data ******************************
000001 /* AUTOMATICALLY GENERATED HEAD */
000002 /*PROGRAMER:TE02   DATE:4 OCT 2011*/
000003 TEST2
000004 TEST3
****** **************************** Bottom of Data ****************************


Command ===>                                                  Scroll ===> CSR
```

图 11-8　编辑宏运行效果

11.12　实验参考答案

1. 基础实验一

Math1 答案

```
/********************REXX***********************/
 SAY 'PLEASE ENTER TWO NUMBER:'
 PULL NUM1 NUM2
 SAY 'YOU ENTER ' NUM1 ' AND 'NUM2
 SAY NUM1'+'NUM2'='NUM1+NUM2
 SAY NUM1'-'NUM2'='NUM1-NUM2
 SAY NUM1'*'NUM2'='NUM1*NUM2
 SAY NUM1'/'NUM2'='NUM1/NUM2
 SAY NUM1 'DIVIDED BY 'NUM2 IS NUM1%NUM2 'REMINDER IS' NUM1//NUM2
```

2. 基础实验二

Math2 答案

```
/********************REXX***********************/
ARG NUM1 NUM2
SAY 'YOU ENTER ' NUM1 ' AND 'NUM2
```

```
SAY NUM1'+'NUM2'='NUM1+NUM2
SAY NUM1'-'NUM2'='NUM1-NUM2
SAY NUM1'*'NUM2'='NUM1*NUM2
IF  NUM2==0 THEN
    SAY 'THE SECOND NUMBER IS 0;DIVISION CANNOT BE DONE'
ELSE
    DO
      SAY NUM1'/'NUM2'='NUM1/NUM2
      SAY NUM1 'DIVIDED BY 'NUM2 IS NUM1%NUM2 'REMINDER IS' NUM1//NUM2
    END
```

3. 基础实验三

Math3 答案

```
/********************REXX**********************/
DO WHILE((DATATYPE(NUM1, 'N' ))<>1|(DATATYPE(NUM2,'N')<>1))
  IF NUM1=='EXIT'THEN
  DO
    SAY'GoodBye'
    EXIT
  END
  SAY 'PLEASE ENTER TWO NUMBER:'
  PULL NUM1 NUM2
END
SAY 'YOU ENTER' NUM1 'AND' NUM2
SAY NUM1'+'NUM2'='NUM1+NUM2
SAY NUM1'-'NUM2'='NUM1-NUM2
SAY NUM1'*'NUM2'='NUM1*NUM2
IF NUM2==0 THEN
  SAY 'THE SECOND NUMBER IS 0;DIVISION CANNOT BE DONE'
ELSE
  DO
    SAY NUM1'/'NUM2'='NUM1/NUM2
    SAY NUM1 'DIVIDED BY 'NUM2 IS NUM1%NUM2 'REMINDER IS' NUM1//NUM2
  END
```

4. 函数和子例程实验

11.5 答案

```
/********************REXX**********************/
SAY 'PLEASE ENTER THE NUMBER:'
PULL NUMBER
DO N=1 TO NUMBER
   SAY 'THE FACTORIAL OF' N 'FROM FUNCTION   IS:' FACTFUNC(N)
   CALL FACTROUTINE N
   SAY 'THE FACTORIAL OF' N 'FROM SUBROUTINE IS:' RESULT
```

```
END
RETURN
/*************FUNCTION************/
  FACTFUNC : PROCEDURE
     N = ARG(1)
     IF N = 1 THEN
        RETURN 1
     RETURN N * FACTFUNC( N - 1 )
/*************SUBROUTINE**********/
  FACTROUTINE: PROCEDURE
     ARG NUM
     S=1
     DO I=1 TO NUM
        S=S*I
     END
     RETURN S
```

5. 执行宿主命令实验

11.8 答案

```
/********************REXX***********************/
SAY 'DO YOU KNOW YOUR PF KEYS?'
PULL ANSWER .
IF ANSWER = 'NO' | ANSWER = 'N' THEN
    ADDRESS ISPEXEC "DISPLAY PANEL(ISPOPT3C)"
ELSE
    SAY 'O.K. NEVER MIND.'
```

6. REXX 构建并提交 JCL 作业实验

11.9 实验

```
/********************REXX***********************/
user = USERID()                        /* obtain user ident    */
queue "//"user"A JOB ACCTR,NOTIFY="user",MSGCLASS=A"
queue "//REXX     EXEC PGM=IKJEFT01, "
queue "//         DYNAMNBR=30,REGION=4096K"
queue "//SYSTSPRT DD SYSOUT=*"
queue "//SYSEXEC  DD DSN="user".REXX.LAB,DISP=SHR"
queue "//SYSTSIN  DD *"
queue "%math2 25 5"
queue "//"
queue "$$"
o = OUTTRAP("output.",,"CONCAT")
ADDRESS TSO "SUBMIT * END($$)"      /* submit jcl job */
o= OUTTRAP(OFF)
SAY output.2
```

```
RETURN
```

7. REXX 调用 ISPF 服务实验

11.10 实验

```
/*******************REXX**********************/
TRACE E
SAY 'ENTER THE SOURCE DATASET AND ITS MEMBER TO COPY'
PULL  FROMDS FROMMEM
SAY 'ENTER THE DESTINATION DATASET AND ITS MEMBER'
PULL  TODS TOMEM
ADDRESS ISPEXEC
'LMINIT DATAID('DDVAR1') DATASET('''FROMDS''') ENQ(SHRW)'
'LMOPEN DATAID('DDVAR1') OPTION(INPUT) LRECL('DLVAR')
        RECFM('RFVAR') ORG('ORGVAR')'
'LMINIT DATAID('DDVAR2') DATASET('''TODS''') ENQ(SHRW)'
'LMCOPY FROMID('DDVAR1') FROMMEM('FROMMEM') TODATAID('DDVAR2')
       TOMEM('TOMEM')'
 IF RC = 0 THEN
    SAY 'COPYING MEMBER COMPLETED.'
 ELSE
     SAY 'COPYING MEMBER FAILED.'
'LMCLOSE DATAID('DDVAR1')'
RETURN
```

8. ISPF 编辑宏实验

REEDIT 实验

```
/*******************REXX**********************/
ADDRESS ISPEXEC
'ISREDIT MACRO'
/*DELETE ALL LINES WITH '*' IN COL1*/
'ISREDIT RESET EXCLUDED '
"ISREDIT EXCLUDE ALL '*' 1"
'ISREDIT DELETE ALL EXCLUDED'  /* DELETE ALL LINES LEFT EXCLUDED*/
/*PUT THE INFORMATION IN HEAD*/
USER=USERID()
CURRENT_DATE=DATE()
"ISREDIT LINE_BEFORE 1 = '/* AUTOMATICALLY GENERATED HEAD */'"
"ISREDIT LINE_AFTER 1 = '/*PROGRAMER:"USER"   DATE:"CURRENT_DATE"*/'"
RETURN
```